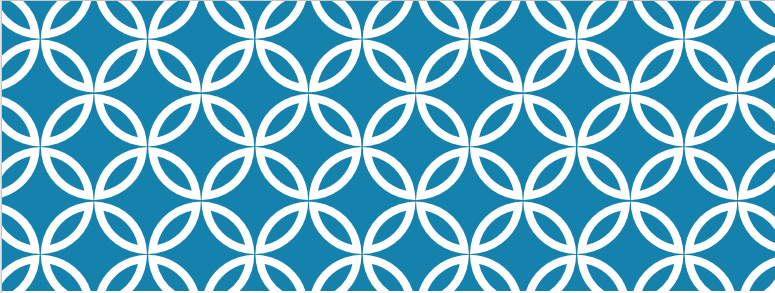
a

Version 1.2| 28 Sep 2018



Master Test Strategy

MOSIP

Modular Open Source Identity Platform

Copyright Information

This document is the exclusive property of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; the recipient agrees that they may not copy, transmit, use or disclose the confidential and proprietary information in this document by any means without the expressed and written consent of Mindtree. By accepting a copy, the recipient agrees to adhere to these conditions to the confidentiality of \_\_\_\_\_\_\_\_\_\_\_\_\_ practices and procedures; and to use these documents solely for responding to \_\_\_\_\_\_\_\_\_\_\_\_\_\_ operations methodology.

Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver** | **Change Description** | **Sections** | **Date** | **Author** | **Reviewer** |
| 0.1 | Initial Draft | All | 24/4/2018 | Prabhashankar Krishnamurthy (PS) |  |
| 0.2 | Internal Review Done | All | 25/4/2018 | PS | Vivek Mathur, Avinash Chandrashekar, Nilesh Dasharathi |
| 0.3 | Internal Review | All | 26/4/2018 | PS | RK, Kalpana |
| 0.4 | Internal update, | Identity Management, Entry Exit Criteria | 5/18/2018 | PS & Nilesh | Vivek |
| 1.0 | Internal Review | All | 5/18/2018 | PS | RK |
| 1.1 | Internal Review | Metrics, Environment, tools | 5/23/2018 | PS, Nilesh, Vishnu | Rekha N |
| 1.2 | Internal Review | Update on Approach and Tools. | 09/28/2018 | Rekha & Avinash | Vivek & Romila |

Reference:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Document Name** | **Version** | **Date** | **Path** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 | MOSIP\_UserStoryBacklog\_Tracker\_ |  |  |  |
| 6 | MOSIP\_Tools\_and\_Technologies |  |  |  |

**Glossary**

|  |  |  |
| --- | --- | --- |
| **Terminology** | **Definition** | **Remarks** |
|  |  |  |
| ABIS | Automated Biometric Identification System |  |
| ara | Arabic | Language code |
| BPOs | Business Process Outsourcing |  |
| CK | Core Kernel |  |
| DMS | Document Management System |  |
| fra | French | Language code |
| HSM | Hardware Security Module |  |
| REGISTRATION PROCESSOR | Identity Management System |  |
| IDP | Identity Platforms |  |
| MA | Morocco | State/Province/ Sub-division code |
| MAR | Morocco | Country code |
| MDM | Master Data Management |  |
| NFR | Non-functional requirements |  |
| OSI | Open Source Initiative |  |
| PKI | Public Key Infrastructure |  |
| PT | Performance testing |  |
| RFC | Request For Comments |  |
| SDK | Software Development Kit |  |
| TSPs | Trusted Service Providers |  |
| UIN | Unique Identification Number |  |
| VPC | Virtual Private Cloud |  |

Table of Contents

[Copyright Information 2](#_Toc528768070)

[Table of Contents 4](#_Toc528768071)

[1. Introduction 9](#_Toc528768072)

[1.1 Purpose 9](#_Toc528768073)

[1.2 Project Overview 9](#_Toc528768074)

[2. MOSIP Test Approach 9](#_Toc528768075)

[2.1 Overall Test Strategy 10](#_Toc528768076)

[2.2 Testing Interfaces for each module 11](#_Toc528768077)

[2.3 Functional Testing 11](#_Toc528768078)

[2.4 Regression Testing 12](#_Toc528768079)

[2.5 Automation Testing 12](#_Toc528768080)

[2.6 NFR testing: 12](#_Toc528768081)

[3. QA Environment & Evaluation Process 13](#_Toc528768082)

[3.1 QA Environment 13](#_Toc528768083)

[3.2 Staging/ Demo Environment 13](#_Toc528768084)

[3.3 NFR Environment: 13](#_Toc528768085)

[4. Test Data Management 14](#_Toc528768086)

[4.1 Test Data Management Process 14](#_Toc528768087)

[5. Strategy to test MOSIP Functionality 14](#_Toc528768088)

[5.1 Kernel 14](#_Toc528768089)

[UIN Generator - API validation for functional and performance 14](#_Toc528768090)

[ID Manager 15](#_Toc528768091)

[OTP Manager 16](#_Toc528768092)

[FTP Manager 16](#_Toc528768093)

[Key Management 17](#_Toc528768094)

[Notification Manager – 17](#_Toc528768095)

[**The test strategy for this component will be updated once the design document is ready** 17](#_Toc528768096)

[PDF Generator 17](#_Toc528768097)

[Configuration Manager – 18](#_Toc528768098)

[**The test strategy for this component will be updated once the design document is ready** 18](#_Toc528768099)

[Data Validator – 18](#_Toc528768100)

[**The test strategy for this component will be updated once the design document is ready** 18](#_Toc528768101)

[Authentication / Authorization 18](#_Toc528768102)

[**The test strategy for this component will be updated once the design document is ready** 18](#_Toc528768103)

[Encryption / Decryption 18](#_Toc528768104)

[**The test strategy for this component will be updated once the design document is ready** 18](#_Toc528768105)

[Zip / Unzip 19](#_Toc528768106)

[5.2 Pre-Registration 19](#_Toc528768107)

[5.1.1 Objective 19](#_Toc528768108)

[5.1.2 Main features and Validation points 20](#_Toc528768109)

[5.1.3 Functionalities to test for Pre-Registration Client 20](#_Toc528768110)

[5.1.4 Test Approach to validate Pre-Registration Client 21](#_Toc528768111)

[Functional verification 21](#_Toc528768112)

[Automation Strategy for Pre- Registration Client: 23](#_Toc528768113)

[Functional Automation – Web App / RWD based design 23](#_Toc528768114)

[Details of testing for Functional Web Automation 23](#_Toc528768115)

[Restful API Automation 25](#_Toc528768116)

[Details of testing for API/Micro services Automation 25](#_Toc528768117)

[Performance Testing 26](#_Toc528768118)

[Web application / RWD design Performance Testing 28](#_Toc528768119)

[5.3 Registration: 30](#_Toc528768120)

[5.2.1 Objective 30](#_Toc528768121)

[5.2.2 Main Features and Validation 30](#_Toc528768122)

[5.2.3 Functionalities to test for Registration Client 30](#_Toc528768123)

[5.2.4 Test Approach in validating Registration Client 31](#_Toc528768124)

[Functional verification 31](#_Toc528768125)

[Automation Testing – Registration Client 35](#_Toc528768126)

[Functional Automation – Desktop app 35](#_Toc528768127)

[Testing of API libraries 37](#_Toc528768128)

[Performance Testing – To be Updated 38](#_Toc528768129)

[5.4 Registration Processor 38](#_Toc528768130)

[5.3.1 Objective 38](#_Toc528768131)

[5.3.2 Main features of Registration Processor are 38](#_Toc528768132)

[5.3.3 Functionalities to test for Registration Processor 39](#_Toc528768133)

[5.3.4 Test Approach to validate Registration Processor 39](#_Toc528768134)

[Functional verification 39](#_Toc528768135)

[Automation Strategy for Registration Processor 41](#_Toc528768136)

[Performance Testing – To be updated 43](#_Toc528768137)

[5.5 Identity Services / Authentication - IDA 43](#_Toc528768138)

[5.4.1 Objective 43](#_Toc528768139)

[5.4.2 Main features of IDA 44](#_Toc528768140)

[5.4.3 Functionalities to test for IDA 44](#_Toc528768141)

[Test Approach in validating IDA 45](#_Toc528768142)

[Functional verification 45](#_Toc528768143)

[5.4.4 Performance Testing 48](#_Toc528768144)

[5.6 Admin (Pre-registration, Registration and REGISTRATION PROCESSOR): 50](#_Toc528768145)

[5.5.1 Objective: 50](#_Toc528768146)

[5.5.2 Pre-Registration Client Admin: 50](#_Toc528768147)

[5.5.3 Registration Admin Functionalities: 51](#_Toc528768148)

[5.5.4 REGISTRATION PROCESSOR Admin functionalities 51](#_Toc528768149)

[5.7 Security Test Strategy 51](#_Toc528768150)

[5.6.1 Secure Code Review 52](#_Toc528768151)

[5.6.2 Web Application Security Assessment 55](#_Toc528768152)

[5.6.3 Web Service Security Assessment 57](#_Toc528768153)

[5.6.4 Thick Client Security Assessment 59](#_Toc528768154)

[6. Functionalities Automatable and Not Automatable 61](#_Toc528768155)

[7. Assumptions & Dependency 62](#_Toc528768156)

[8. Defect Management 64](#_Toc528768157)

[9. Testing Tool Stack 66](#_Toc528768158)

[10. Quality Gates 68](#_Toc528768159)

[10.1 Entry Criteria 68](#_Toc528768163)

[10.2 Exit Criteria 68](#_Toc528768164)

[10.3 Suspension Criteria 68](#_Toc528768165)

[10.4 Resumption Criteria 68](#_Toc528768166)

[11. RASCI matrix 69](#_Toc528768167)

[12. Test Deliverables 71](#_Toc528768168)

[13. Test metrics 71](#_Toc528768169)

[13.1 Functional Metrics 71](#_Toc528768170)

[13.2 Performance Metrics 73](#_Toc528768171)

[13.3 Security Testing Metrics 74](#_Toc528768172)

Figure

[Figure 1: Conceptual Model 10](#_Toc514436658)

[Figure 2 : Testing Scope – Birds Eye View 11](#_Toc514436659)

[Figure 3 : MOSIP Validation Approach 12](#_Toc514436660)

[Figure 4 : QA Environment & Evaluation Process 14](#_Toc514436661)

[Figure 5 : Pre Registration Client Testing 16](#_Toc514436662)

[Figure 6 : API/ Micro Services Automation 19](#_Toc514436663)

[Figure 7 :Registration Client Testing 26](#_Toc514436664)

[Figure 9: API Micro Service Automation 29](#_Toc514436665)

[Figure 10 : Functional Automation of thick client 30](#_Toc514436666)

[Figure 10 : REGISTRATION PROCESSOR Testing 32](#_Toc514436667)

[Figure 12 : API / Micro Services Automation 34](#_Toc514436668)

[Figure 12 : Identity Service Testing 38](#_Toc514436669)

[Figure 14 : Web Application security Assessment 46](#_Toc514436670)

[Figure 15 : Web Security Assessment 48](#_Toc514436671)

[Figure 16 : Defect Cycle 55](#_Toc514436672)

# Introduction

## Purpose

The purpose of this document is to define the high-level test strategy for **M**odular **O**pen **S**ource **I**dentity **P**latform (**MOSIP**) defining the preliminary test scope, high-level test activities followed for quality delivery. MOSIP functionalities will be validated at individual component level and at end-to-end level. The test strategy provides the framework for governing the testing activities performed for the project. It also serves as a reference in describing the test approach, tools, environments, roles, responsibilities, and processes required for qualifying the MOSIP program.

## Project Overview

Modular Open Source Identity Platform (MOSIP) is a Digital Identity system implemented with a vision to enable any country to provision Unique Digital Identity Number (UDIN) for their resident and nonresident citizens. Key stakeholders for this platform implementation would like to implement the system as an Open Source Project, with a community of users, contributors and developers.

# MOSIP Test Approach

Based on the needs of MOSIP, a well-defined structured testing approach is designed to carry out comprehensive testing. The test methodology describes various testing types applicable at different phases that will be covered. Test methodology details the following key aspects:

* Digital Domain best practices
* Re-usability of components and test cases
* Increased automation to bring in efficiency and effectiveness
* Business stability with early detection of defects
* Testing across devices and multiple browsers to increase end user experience
* Agile Methodology

Following are the modules considered for testing of MOSIP

* Pre-Registration
* Registration
* Registration Processor
* IDA
* Admin application (Pre-Registration, Registration, Registration Processor)

## Overall Test Strategy

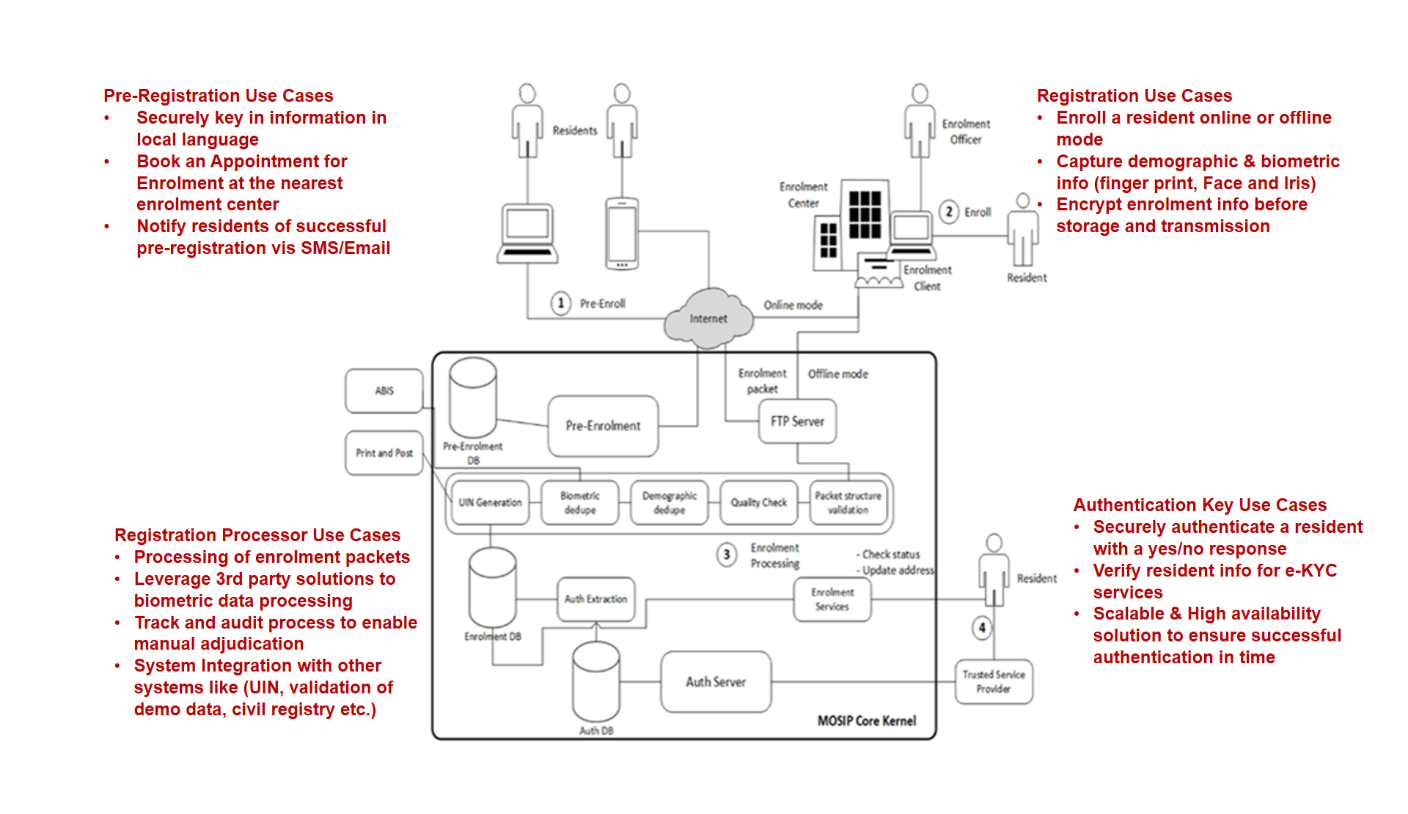


Figure 1: Conceptual Model

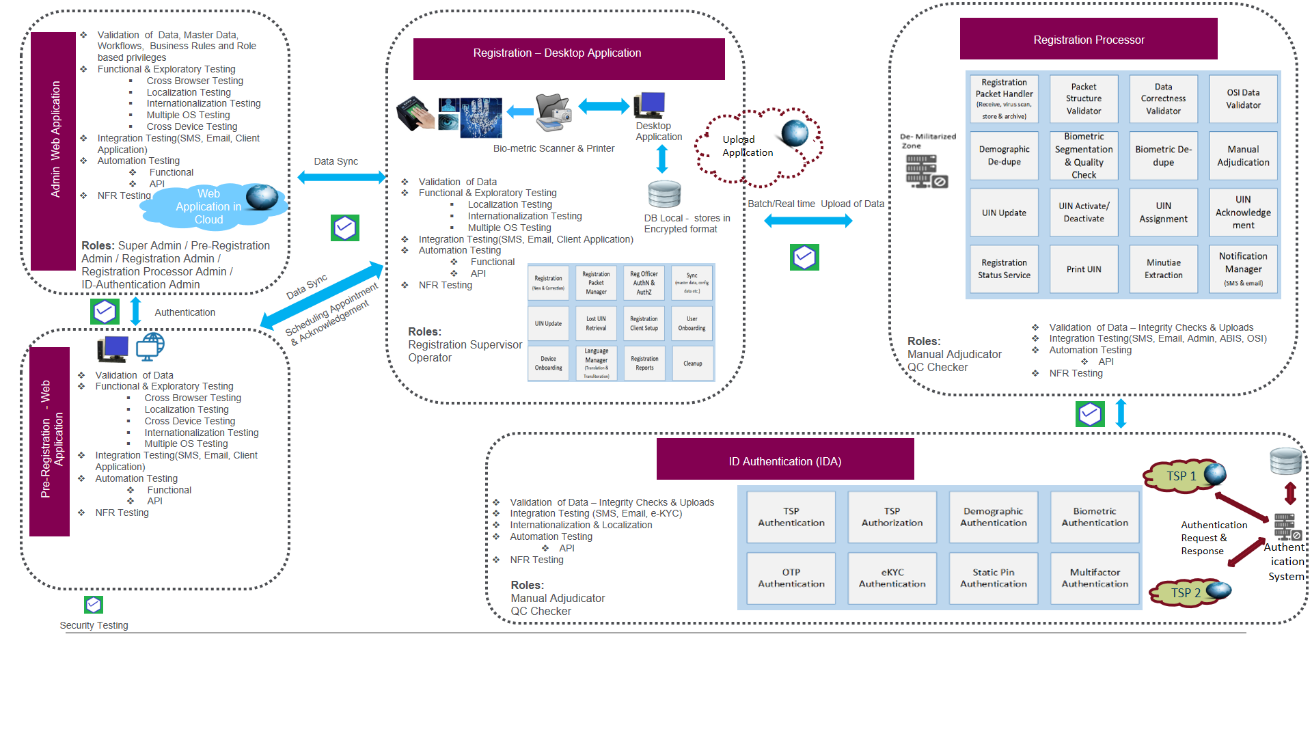


Figure 2 : Testing Scope – Birds Eye View

The above diagram, Figure 2, details out the functionalities considered for testing both from component level as well at the holistic level. The validation approach is defined in such a way that its in-line with the program level delivery model.

In MOSIP, test scenarios are covered using **Visual Test Design approach using the Mind Map**. To validate MOSIP both functional and non-functional mode following techniques are implied .

Figure 3 : MOSIP Validation Approach

## Testing Interfaces for each module

|  |  |  |  |
| --- | --- | --- | --- |
| **Modules** | **Java API %** | **Restful API %** | **UI %** |
| **Pre-Registration Module** |  | 20% | 80% |
| **Registration Module** | 25% |  | 75% |
| **Registration Processor** | 20% | 80% |  |
| **ID - Authentication** |  | 100% |  |
| **Kernel** | 60% | 40% |  |

## Functional Testing

Testing the application from the functional perspective both from component level as well from an end-to-end perspective in adherence to the defined business rules and requirements.

## Regression Testing

To ensure that the defined functionality is not changed. Optimal regression suite is created which provides maximum test coverage with minimal risk and less effort to maintain and execute.

The regression-testing suite should have Risk based; Impact based test scenarios along with prioritization of functionalities based on the changes and end-to-end scenarios. Functionalities affected due to the defect fixes are also regressed. All CRs implications on the affected modules are also part of Regression suite

## Automation Testing

A faster test cycle at lower costs is essential, re-usable test automation frameworks coupled with open-source tools and technologies is a key solution to shrink test cycle times.

As part of Automation strategy for MOSIP a robust Test Automation will be derived, which offers a complete solution to ensure that all the automation objectives are met.

For MOSIP, Test automation approach would cover the following aspects

* Regression Testing - New feature, bug fixes, enhancement testing
* Cross-browser testing – Web applications
* Mobile, Tablet device testing (iOS, Android) – Web application
* End to End Test Suite – Automated business critical functions
* Continuous Integration – Automated tests with every build
* Web services testing – Restful API
* Compatibility testing

## NFR testing:

NFR testing on MOSIP is essential to ensure all the non-functional requirements are working as per the define requirements.

Following techniques are considered at high level for validating NFRs for MOSIP

* Performance Testing: Is to ensure and measure the application’s performance at varied scenarios against defined NFRs for example: resource utilization, throughput etc…
* Endurance Testing: where varied scenarios are tested by putting constant load for varied amount of time, and validate the system behaviors against each scenario.
* Security Testing: Robust security architecture is essential and provides a roadmap to implementing necessary data protection measures to avoid data breach and compliance related issues. The goal is to manage the private data appropriately and take every precaution to protect it from loss, unauthorized access or theft.

The security activities that will be carried out for identification of security issues as per applicability are as below:

* + Secure Code review – All Modules
  + Web Application Security Assessment – Pre-Registration, Registration Processor, Admin applications
  + API Security Assessment – Registration, Registration Processor, IDA
  + Thick Client Security Assessment – Registration

# QA Environment & Evaluation Process

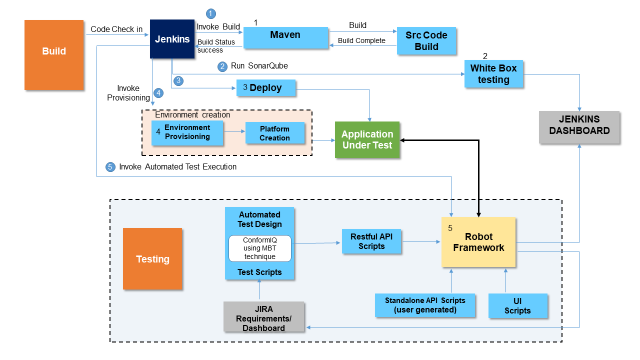


Figure 4 : QA Environment & Evaluation Process

The diagram above details on how the environments to be used for MOSIP evaluation, this also details out on the frequency of delivery of application for testing.

## QA Environment

Once code gets checked in and build gets automatically deployed on QA environment. Automated smoke test script will be run on the new build to validate the build stability. If the build is stable and there are no regression issues, further testing be implied on the build.

## Staging/ Demo Environment

Post qualifying a milestone or a release the code base pertaining to that release will be moved to the Staging or Demo environment. Where the customer can view the application features as part of the demo or the same can be used for verifying the functionality as part of UAT.

## NFR Environment:

Deploy post the code is functionally tested and deployed on the Staging or Demo environment, the same code base on to the NFR environment for NFR specific testing.

# Test Data Management

QA will have a “Test Data Management” which will address the end-to-end test data management in a holistic manner.

## Test Data Management Process

Testing team will have Test Data created using utilities that will be based on the requirement and defined formats. We believe this holistic approach will allow the entire eco system around the test data management to mature and become efficient.

The test data will be simulated to generate data, which can further influence the results. The test data will be validated and the resulting test cases. Reusability is key for ensuring cost-effectiveness and maximizing the testing efforts. Test Data will be built and segmented to make it more and more reusable and the objective is to use it as much as possible and optimize the value of work that has been done.

Test Data Management entails scripting, data generation, cloning, and provisioning. Automation or writing scripts / programs for all these activities will accelerate the process and efficiency. The following are identified as part of Test Data Management:

* Alphanumeric Test data generation
* Image Test data generation
* Image Test data generation - use simple techniques like image markings for easy operations
* Build a dummy ABIS using simple image markings
* Test data organization and storage

# Strategy to test MOSIP Functionality

## Kernel

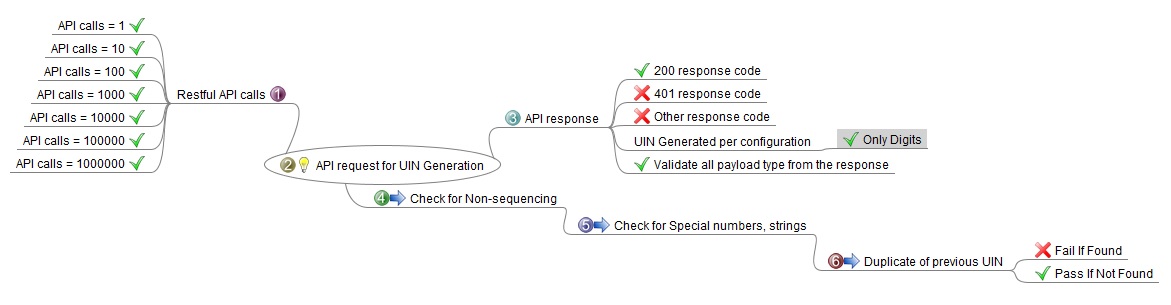
The Kernel module would comprise of the following Java API services and below is the approach used to test each component present in the kernel module:

|  |  |  |
| --- | --- | --- |
| UIN Generator | Key Management | Data Validator |
| ID Manager | Notification Manager | Authentication / Authorization |
| OTP Manager | PDF Generator | Encryption / Decryption |
| FTP Manager | Configuration Manager (XSD) | Zip / Unzip |

### UIN Generator - API validation for functional and performance

This modular API will generate a unique identification number. The Test scenarios would check for uniqueness of the string/number generated and would have checkpoints to test for all the combination of scenario for e.g. checking non-sequencing, special and easily guessable number.

UIN Generator will be validated thoroughly using a Java program with many combinations of data as described below



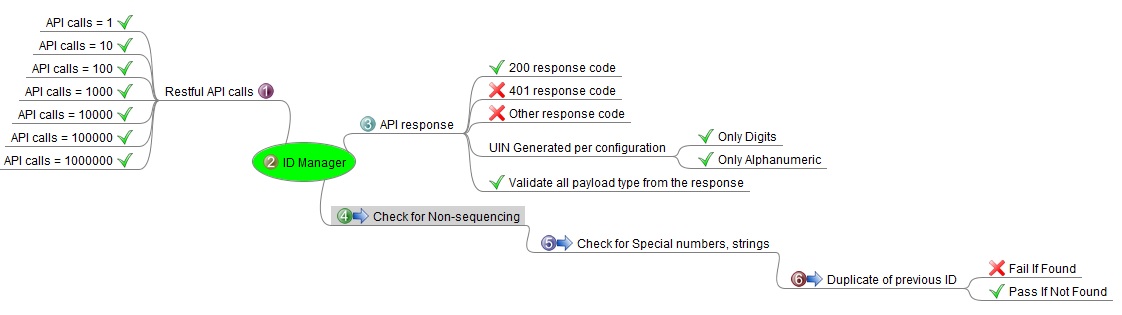
### ID Manager

The ID manager comprises of utilities/functions which shall be called by the respective modules (Pre-registration; Registration; Registration processor; ID Authentication) for generation of single ID generation request which comprises of either one or more as described below

* RID Generator
* PRID Generator
* VID Generator
* Token ID Generator
* Static PIN Generator
* Registration Center ID Generator
* Internal User ID Generator
* TSP ID Generator

This modular API will generate an identification number and could be based on integer’s data set or string data set or a combination of both. The Test scenarios would check for the uniqueness of the string/number generated for the respective ID generation request and would have checkpoints to test for all the combination of scenario.

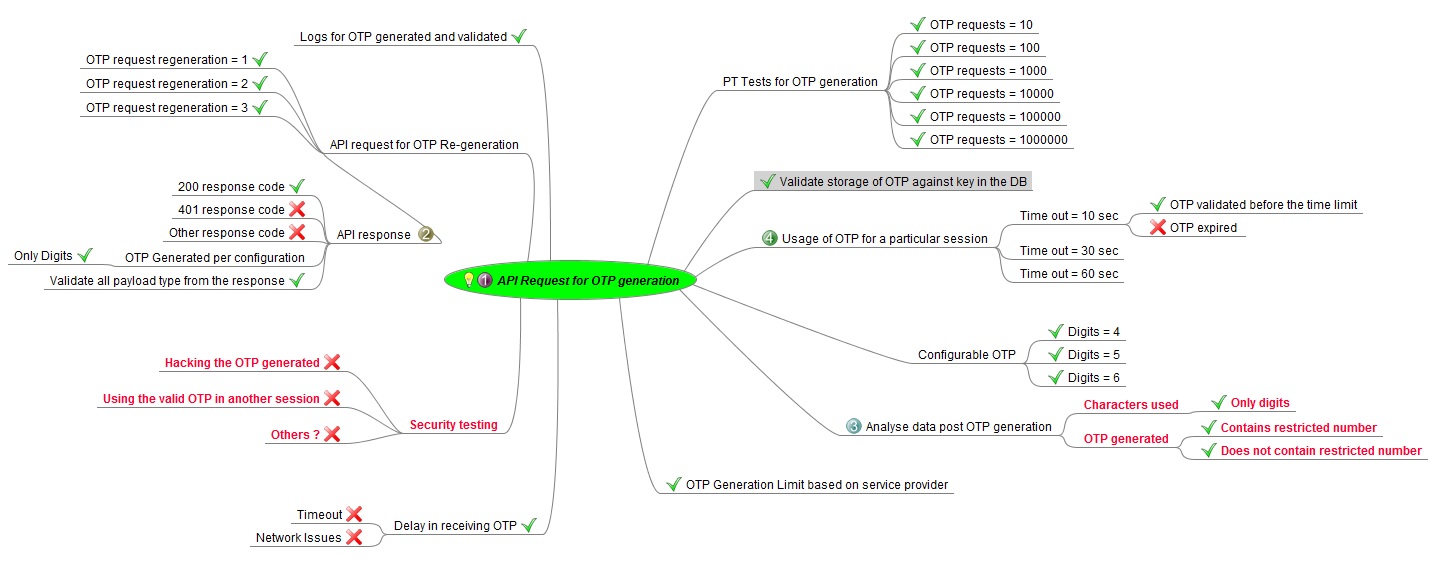
ID Manager will be validated thoroughly using a Java program with many combinations of data; one of the sample is as described below



### OTP Manager

This modular API will generate an OTP number based on a certain algorithm. The current test strategy for OTP generation will be on the combination of timestamp and a string.

OTP generator and validator API will be tested through a Restful API request message and a Java program utility as a validation engine as shown in the below mindmap diagram.



### FTP Manager

Functional Testing

For functional testing of the FTP manager we shall be using the SSH library and develop Java utilities to validate the files and folders, their file sizes before transfer and after transfer.



### Key Management

The Key management modular API will generate the synchronous and asynchronous keys based on the input parameters requested by various modules. For e.g. the registration module for encryption of the packet will call this modular API to generate the keys which will be active for a set number of time duration (private and public); the key manager generates the keys per the said algorithm type.

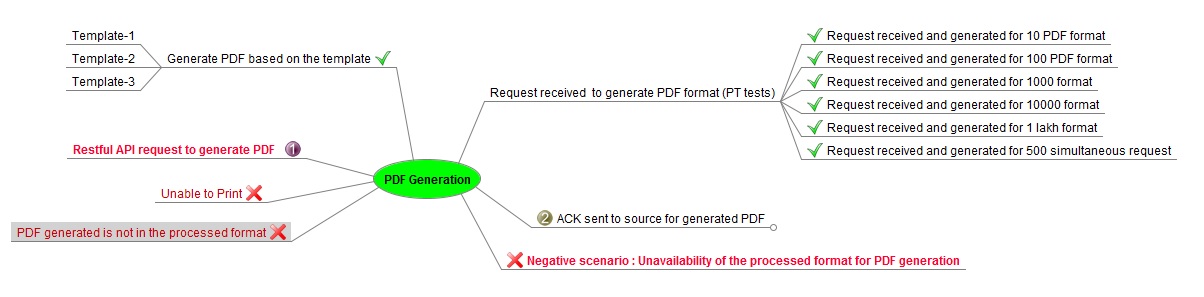


### Notification Manager –

### **The test strategy for this component will be updated once the design document is ready**

### PDF Generator

The PDF generator modular API will generate the PDF document/files based on the specific templates. For validation purpose a True copy (in image) format for all the templates will be maintained in the repository. The automated script will call this API and the output document generated will be compared with the True copy for its right format presented and will used a custom library to read/validate in the data parameters for the generated template as depicted in the below mindmap diagram



### Configuration Manager –

### **The test strategy for this component will be updated once the design document is ready**

### Data Validator –

### **The test strategy for this component will be updated once the design document is ready**

### Authentication / Authorization

### **The test strategy for this component will be updated once the design document is ready**

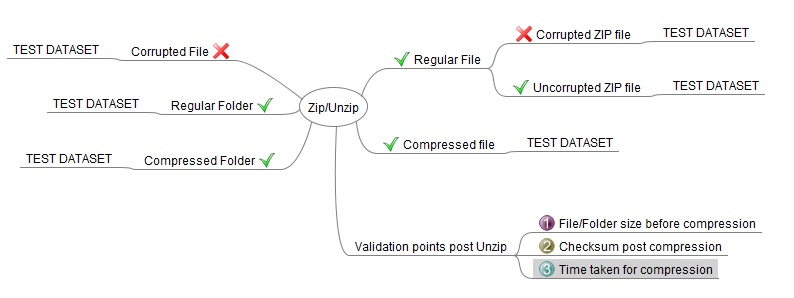
### Encryption / Decryption

### **The test strategy for this component will be updated once the design document is ready**

The encryption and decryption modular API will be tested on a regular file (e.g. json, xml) and folders (e.g. packets) using the key generated from the Keyword manager API. A java program will be developed which shall validate the packet/file post encryption for the change/abstracted data set in the files. If the file/packet is in zipped format, it should not be able to be extracted. The java program will take in multiple combination of packets and read the JSON file attributes and begin decrypting the packet using valid and invalid combination of key which are as described in the mindmap diagram below

### Zip / Unzip

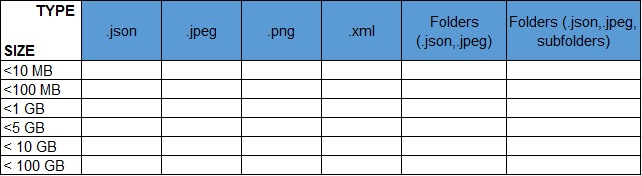
The ZIP and Unzip utility will be tested using manually and automated approach as described in the mindmap diagram below



The automated scripts will send various combination of files (i.e. contents and size) to this modular API to test the ZIP functionality and shall measure the size of the folder/file (before the execution); time/duration taken to execute, the compressed file size and checksum (alphanumeric string).

The Files/Folders will be transferred to a remote system or either in a local system and then the UNZIP API will be executed. The output contents of the UNZIP files/folders will be compared with the original combination(s) of file

Sample TEST DATA SET for the files will be as shown below



## Pre-Registration

### Objective

The objective of the pre-registration portal would be to capture the demographic data, document upload schedule and manage appointment to reduce the registration time at registration centre. This will also help in increasing throughput of registration centre and reduction of overall registration process. The pre-registration service will be offered through the citizen portaland citizen mobile application.

### Main features and Validation points

­­

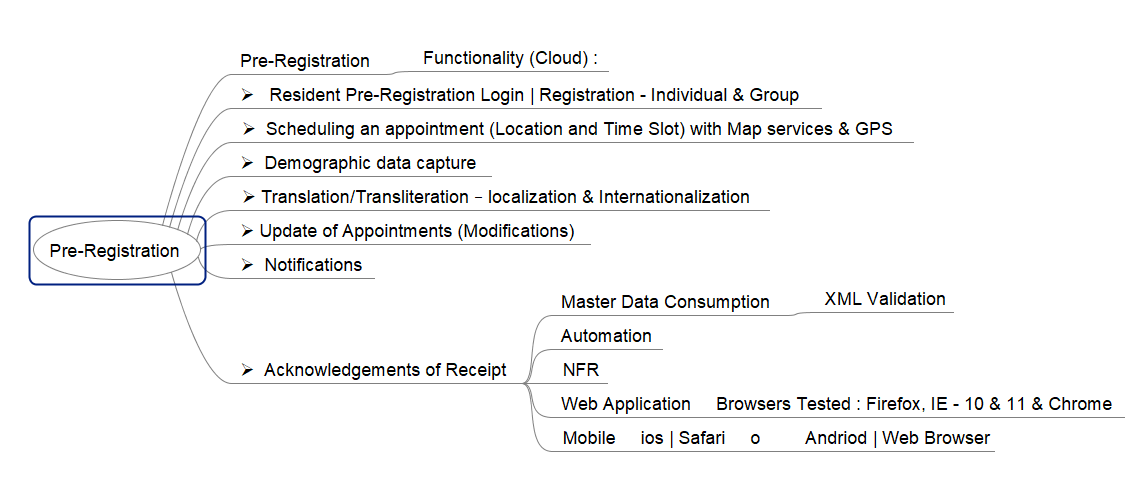
Figure 5 : Pre Registration Client Testing

### Functionalities to test for Pre-Registration Client

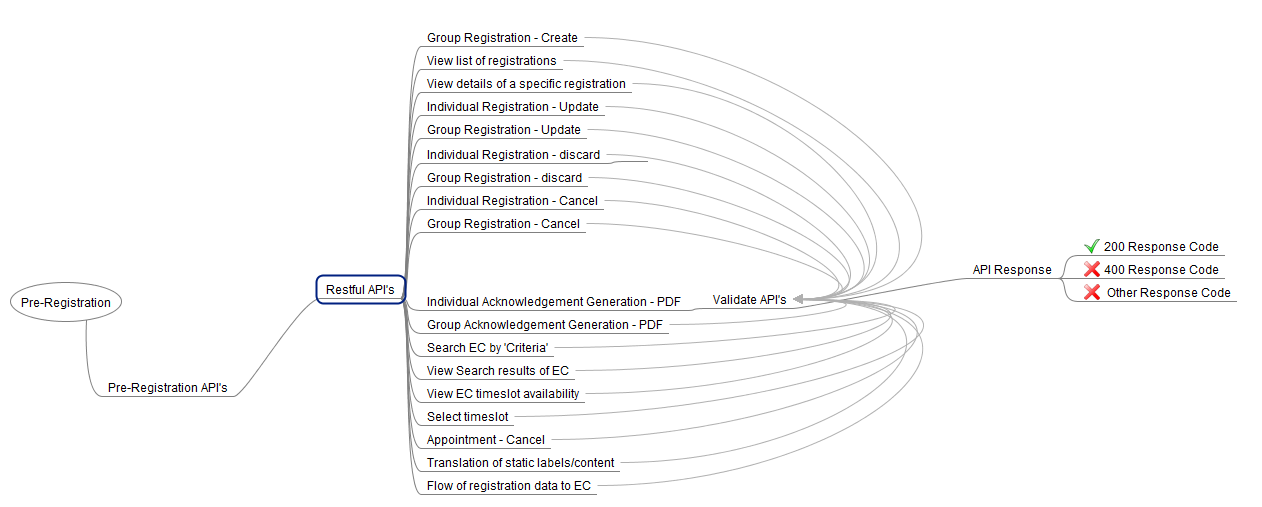
* UI Interface Application – Resident Pre-registration Login
* Scheduling an appointment (Location and Time Slot)
* Map services
* Demographic data capture & Files upload
* Translation/Transliteration
* Update of Appointments (Modifications)
* Notifications
* Acknowledgements of Receipt

### Test Approach to validate Pre-Registration Client

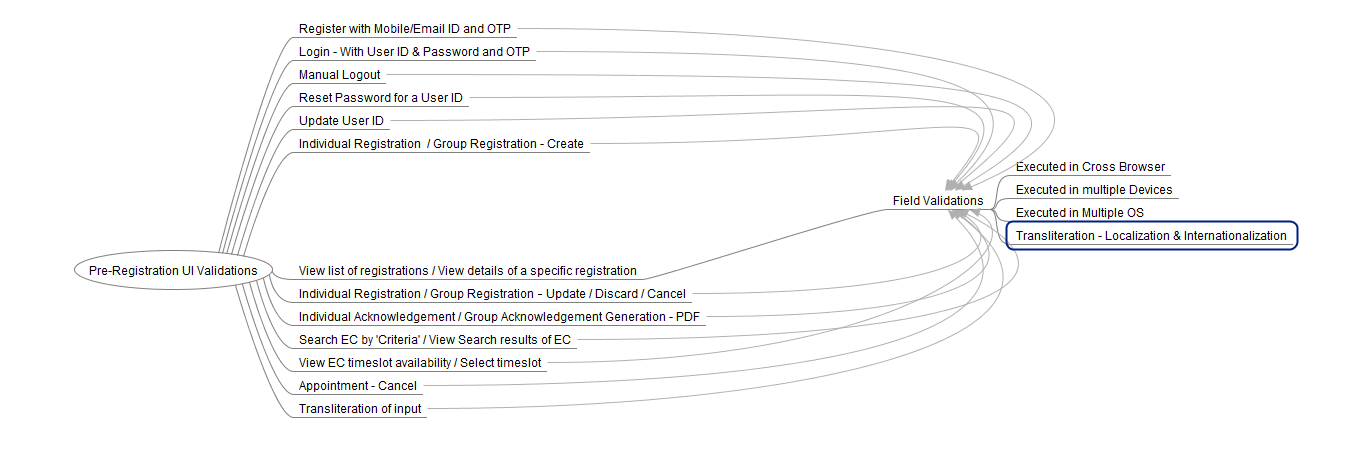
### Functional verification



The below listed API’s will be validated thoroughly using a Java program with many combinations of data.



The below listed UI validations will be tested thoroughly:

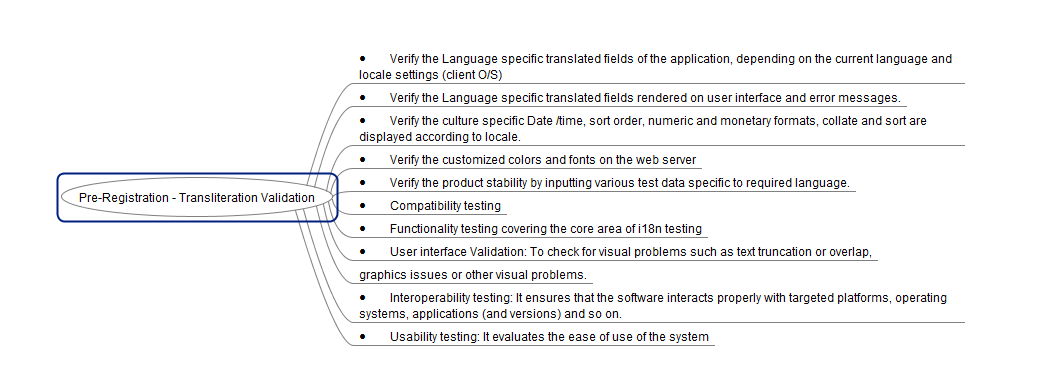


Residents provide demographic details in a language of choice provided by the country.

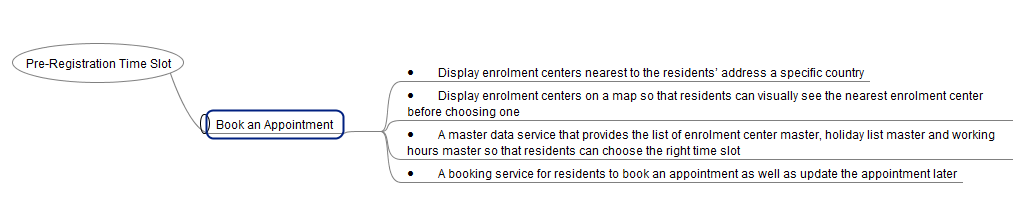
Residents will have a choice to enter the name in either language. Residents edit the value entered to ensure appropriateness.

* A **transliteration service** to help the customer with transliterating the values entered
* A **standard interface** such that any **transliteration library** can be used for a specific country
* Support RTL and LTR character sets.

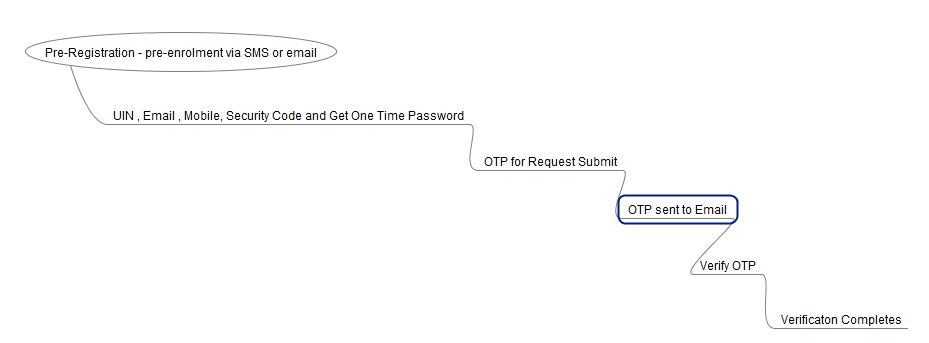
**Transliteration testing covers the following:**



Residents choose an registration center and time slot from the available list of centers and book an appointment for completing the registration procedure.

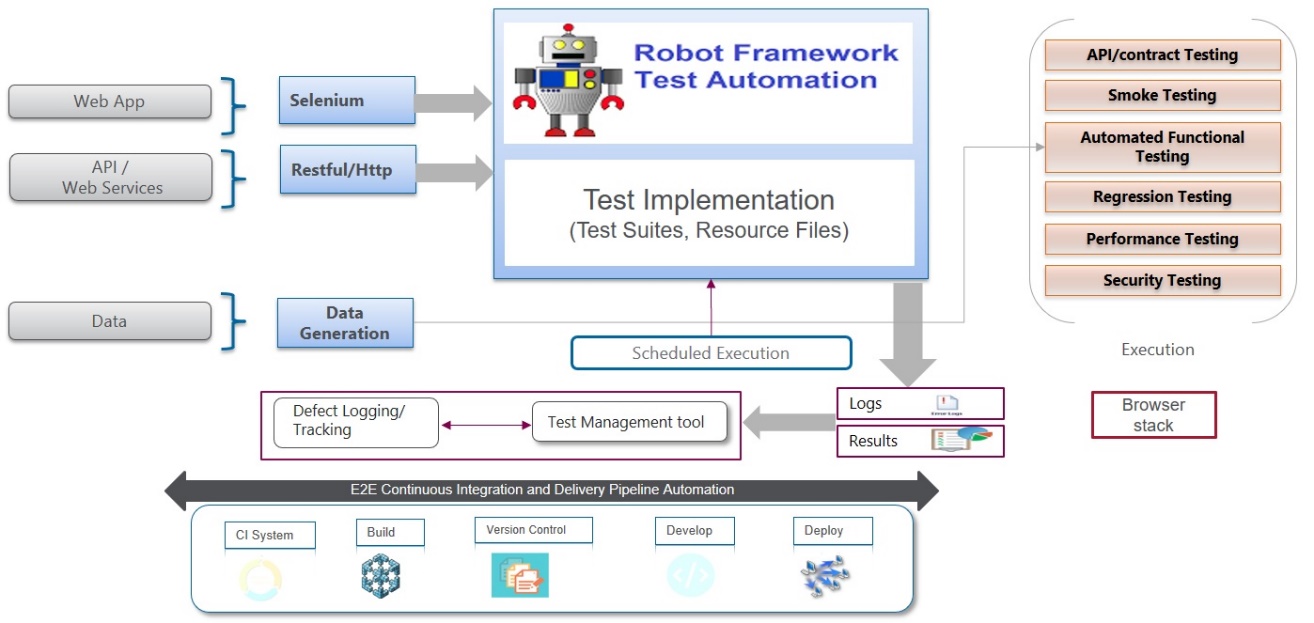


Residents receiving notifications



### Automation Strategy for Pre- Registration Client:

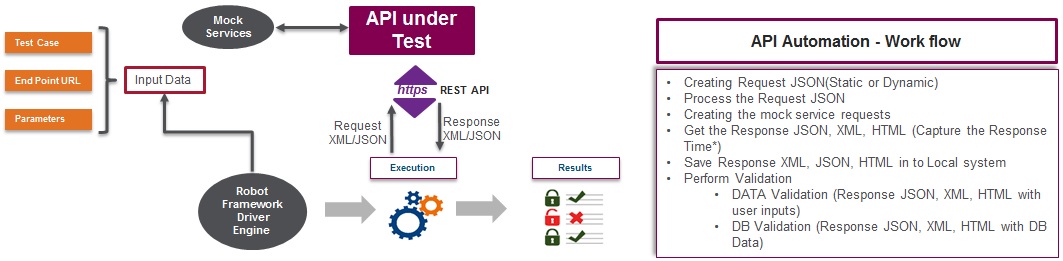
### Functional Automation – Web App / RWD based design



### Details of testing for Functional Web Automation

|  |  |
| --- | --- |
| Objective | Functional automation testing is performed to verify that the defined functionalities / requirements of the Pre-Registration applications are met and the functions conform to the design & architecture specifications in an automated fashion.  Some potential scenarios where functional automation testing can be considered are:   * Accept pre-registration requests from Residents over the web * Residents provide basic demographic information in languages provided by the country * Residents choose a time slot and book appointment for registration * Residents change the pre-registrationdetails or booking * Residents are notified of a successful pre-registrationvia email |
| Preparation Activities | This is an integral part from functional validation perspective which would be the building block considering the complete application. Listed below are the preparation activities which are expected to be performed:   1. Understanding the functionality from an individual module / application perspective as well as from an integrated mode considering the holistic approach 2. Understanding the functional and technical requirements and specification documents 3. Understand functionalities that can be automated 4. Identification of automation test scenarios based on the defined specification document and use cases 5. Perform ROI analysis on the same 6. Walk through the test scenarios with the stakeholders for alignment. 7. Creation of Test Data need for execution 8. Creation of mock service or virtual scripts for components unavailable (i.e. getting the pre-registrationACK unique number, passing the OTP into the system to proceed through the authentication, document is uploaded and to get a positive or negative ACK) 9. Creation of automation test scripts based on the scenarios and then prioritized based on the criticality 10. Create Requirement traceability matrix (RTM) from specification document 🡪 Use case 🡪 Test scenario 🡪 Test cases |
| Execution Approach | The automation tool going to be used will be Selenium library within Robot framework.   1. **Sample Test Definition/Scenario** : where the resident would fill in his demographic details comprising of name, date of birth, gender and address   **Automation Validation Points** :  Pre-defined Test Data will be created for every customer for registration (pre-registration) with relevant valid and invalid combination.  The Automated scripts will validate the text fields as per the configurations per Portal.  For higher Reusability, The Automation Framework will maintain a separate Config file of Characters & Integer set for reading from the input and validate the same with various combination of <TEST DATA> and validate the client UI handling the exception before passing to the details to the Web API   1. **Sample Test Definition/Scenario** : Resident uploads his identity documents   **Automation Validation Points** :  The Automated scripts to validate different type/format of the docs allowed as per the configuration file and seek for a successful ACK from the Mock/API server (from DMS) once upload gets successful.    The contents present within the document won't be validated  The following activities are carried out as part of this activity:   1. Post all the preconditions are met the application under test will be validated against the test scripts defined based on the specification documents. This execution takes the precedence based on the priority. 2. The scenarios would be tested on latest one version of **Chrome** and **Firefox** browsers on Windows and **Mozilla** on Linux PC, one version of Chrome on last two stable versions of Android and one version of **Safari** of last two stable versions of iOS as part of Cross Browser Testing 3. Defect cycles are initiated based on the failed cases as well the bugs detected during the execution. 4. Expectations on the exit criteria are evaluated towards sign off to take call on go – no go decision on the functionality to next phase. |
| Prerequisites | 1. Test environment is available with appropriate data 2. Necessary test data is created for the testing to be executed 3. Mock service scripts are ready and available 4. Test Strategy, Plan & Scenarios are signed off |
| Environment | Test environment |
| Responsibility | Test team (automation testing associates) |
| Other Considerations | The above solution is based on the current understanding of the MOSIP landscape |

### Restful API Automation

****

### Details of testing for API/Micro services Automation

|  |  |
| --- | --- |
| Objective | The objective is to test the business layer logic of the software architecture (i.e. Functional Testing) between   1. User Interface 🡨🡪 Service Endpoints through the API Gateway 2. Service Endpoints 🡨🡪 Kernel Components 3. Service Endpoints 🡨🡪 Integration points 4. Service Endpoints 🡨🡪 Pre-registrationdata   which is been implemented using REST protocol to carry JSON format data over HTTP endpoint |
| Preparation Activities | Web/Micro Services Testing will basically involve   * Determining the operations the web service provides * Determining the JSON request format which the tester needs to send * Determining the response JSON format * Using a tool to send request and validate the response   Web Service Test Automation library like RestInstance or Rest-assured will be used in this project which will be integrated within the Robotframework.  Mock services will be created either using Wiremock or Citrusframework for completing the validation flow in absence of the DB or the related API component |
| Execution Approach | Entire web services testing will be automated. The automation framework will be comprising of Rest-Assured or RestInstance jars for complete testing of the micro services and the API.  For each of the API   1. Understand the functionality of the API program and clearly define the scope of the testing activity 2. Apply testing techniques such as equivalence classes, boundary value analysis and error guessing and write test cases for the API 3. Input Parameters for the API need to be planned and defined appropriately 4. Execute the test cases and compare expected and actual results |
| Prerequisites | 1. Components, Database and server if not available will be attempted to pass-through using a mock script 2. Parameter selection and categorization required to be known to the testers 3. All communication & data exchange will be based on JSON 4. Assumption that there is no need of virtualization environment to be created |
| Environment | Test environment |
| Responsibility | Test team (automation testing associates) |

### Performance Testing

We have considered two main components for performance testing under pre-registrationmodule viz.

1. Web services / micro services – which integrate pre-registrationmodule with other modules in MOSIP
2. Web application - E2E workflows for citizens and admins

Web services / Micro services Performance Testing

|  |  |
| --- | --- |
| Scope | **In Scope**   * Web Services / Micro services Performance Testing   **Out of Scope**   * Notification services (email / SMS) |
| Objective | The objectives of performance testing are   * To determine the **throughput of the services** and also ensuring that there are no bottlenecks in the system. * To determine if web / micro services can scale if additional hardware is provided - (TBD - vertical / horizontal) * To determine if services can withstand continuous usage for xx hours   Note: Actual numbers for each of the objective will be decided later in consultation with architects and business analysts |
| Preparation Activities | Listed below are the preparation activities which are expected to be carried out:   1. Understanding the overall application functionality and understanding business critical features of the application 2. Understanding the functional and technical requirements and specification documents. Use cases can be employed to provide clarity of the desired functionality 3. Identification of services based on the defined specification document and use cases    * Frequently used web services / micro services    * Business critical web services / micro services 4. Walkthrough of the web / micro services with the stakeholders for alignment 5. Create workload models for identified services 6. Prepare test plan including scope, workload model and SLAs / KPIs for performance   Mock services will be created either using *Wiremock* or *Citrus framework* for completing the validation of performance in absence of the DB or the related API component  Note: we will reuse mock services created for test automation purpose |
| Execution Approach | Performance testing focuses on understanding server capacity to handle load in terms of requests. There are different types of testing depending on their purpose, the amount of requests generated, the pattern on load generation and the duration for which load-tests are executed.  Keeping in mind objectives for MOSIP we will carry out following types of testing:   1. **Load Test:** This involves testing the application under normal expected load by way of gradually increasing the load. Web services / micro services will be tested for gradually increasing number of requests.   For example, 0.10x 🡪 0.25x 🡪 0.50x 🡪 1x of expected load  We will create different workload model to simulate real life scenarios.   1. **Scalability Testing:** This involves carrying out load tests by increasing number of requests to understand scalability of application with additional hardware. Purpose of scalability testing is to identify major workloads and mitigate bottlenecks that can impede the scalability of the application to meet the load demands of the coming years, in a graceful manner.   For example – 1.25x load 🡪 1.75x load 🡪 2x load with additional hardware   1. **Endurance Testing –** This involves testing services with sustained load over extended period of time. It helps to find the following kind of bugs:-    1. Resource leak    2. Timing    3. Hardware-related    4. Counter-overflow   For example – 1x load for 8 / 16 / 24 hours  **Server Monitoring**  Key parameters of underlying infrastructure would be monitored while carrying out above-mentioned tests. |
| Prerequisites | 1. Application is functionally stable without any S1/S2 defects 2. Necessary test data is created for the testing to be executed 3. Test Strategy, Plan & Scenarios are signed off 4. Test environment is available with appropriate data 5. Appropriate access permissions are setup for server monitoring and for any other purpose identified in Test strategy / plan 6. Components, Database and server if not available will be attempted to pass-through using a mock script 7. Parameter selection and categorization required to be known to the testers 8. All communication & data exchange will be based on XML/JSON 9. Assumption that there is no need of virtualization environment to be created |
| Sample Scenarios | 1. Registration Centre Search Service – based on area code / area name 2. Appointment booking service 3. Appointment Update service |
| Environment | Test environment (preferably which mimics production environment closely) |
| Responsibility | Test team (performance testing associates) |

### Web application / RWD design Performance Testing

|  |  |
| --- | --- |
| Scope | **In Scope**   * Web application Performance Testing   + Testing for E2E workflows   **Out of Scope**   * Notification services (email / SMS) |
| Objective | The objectives of performance testing are   * To determine the **number of concurrent users** that application can support and also ensuring that there are no bottlenecks in the system. * To determine if web application can scale if additional hardware is provided - (vertical / horizontal) * To determine if application can withstand continuous usage for xx hours   Note: Actual numbers for each of the objective will be decided later in consultation with architects and business analysts |
| Preparation Activities | Listed below are the preparation activities which are expected to be carried out:   1. Understanding the overall application functionality and understanding business critical features of the application 2. Understanding the functional and technical requirements and specification documents. Use cases can be employed to provide clarity of the desired functionality 3. Identification of test scenarios / workflows based on the defined specification document and use cases    * Frequently used workflows    * Business critical workflows 4. Creation of test scenarios / workflows 5. Walkthrough of the workflows with the stakeholders for alignment 6. Create workload models for identified workflows 7. Prepare test plan including scope, workload model and SLAs / KPIs for performance |
| Execution Approach | For the pre-registrationweb application, we will carry out both server side performance testing and client side performance testing.  **Server Side Performance Testing**  Server side Performance testing focuses on understanding server capacity to handle load in terms of number of concurrent users. There are different types of testing depending on their purpose, the amount of requests generated, the pattern on load generation and the duration for which load-tests are executed.  Keeping in mind objectives for MOSIP we will carry out following types of testing:   1. **Load Test:** This involves testing the application under normal expected load by way of gradually increasing the load. Web services / micro services will be tested for gradually increasing number of requests.   For example, 0.10x 🡪 0.25x 🡪 0.50x 🡪 1x of expected load  We will create different workload model to simulate real life scenarios.   1. **Scalability Testing:** This involves carrying out load tests by increasing number of requests to understand scalability of application with additional hardware. Purpose of scalability testing is to identify major workloads and mitigate bottlenecks that can impede the scalability of the application to meet the load demands of the coming years, in a graceful manner.   For example – 1.25x load 🡪 1.75x load 🡪 2x load with additional hardware   1. **Endurance Testing –** This involves testing services with sustained load over extended period of time. It helps to find the following kind of bugs:-    1. Resource leak    2. Timing    3. Hardware-related    4. Counter-overflow   For example – 1x load for 8 / 16 / 24 hours  **Server Monitoring**  Key parameters of underlying infrastructure would be monitored while carrying out above-mentioned tests.  **Client Side Performance Testing**  Client side performance testing focuses on measuring response time experienced by end user. It also involves identifying areas to improve for a web page rendering.  Key activities in this involve:   * Define test matrix across parameters such as bandwidth and browser * Prepare script for identified pages / workflows * Execute multiple iterations of executions to take measurements * Analyse the results and identify areas of improvement for web pages * Prepare report |
| Prerequisites | 1. Application is functionally stable without any S1/S2 defects 2. Necessary test data is created for the testing to be executed 3. Test Strategy, Plan & Scenarios are signed off 4. Test environment is available with appropriate data 5. Appropriate access permissions are setup for server monitoring and for any other purpose identified in Test strategy / plan 6. All communication & data exchange will be based on http/s |
| Sample Scenarios | 1. Citizen searches for registration centre based on area code / area name 2. Citizen fills a form for booking an appointment for registration at an registration centre 3. Citizen updates his appointment at an registration centre |
| Environment | Test environment (preferably which mimics production environment closely) |
| Responsibility | Test team (performance testing associates) |

## Registration:

### Objective

The complete functional evaluation of the registration client would be to undertake registration at the registration centre for both pre-enrolled residents as well the walk-in residents. This client software integrates admin application along with biometric capturing devices, which are compliant to identified biometric standards. Notify post successful registration process to the resident with an acknowledgement.

### Main Features and Validation

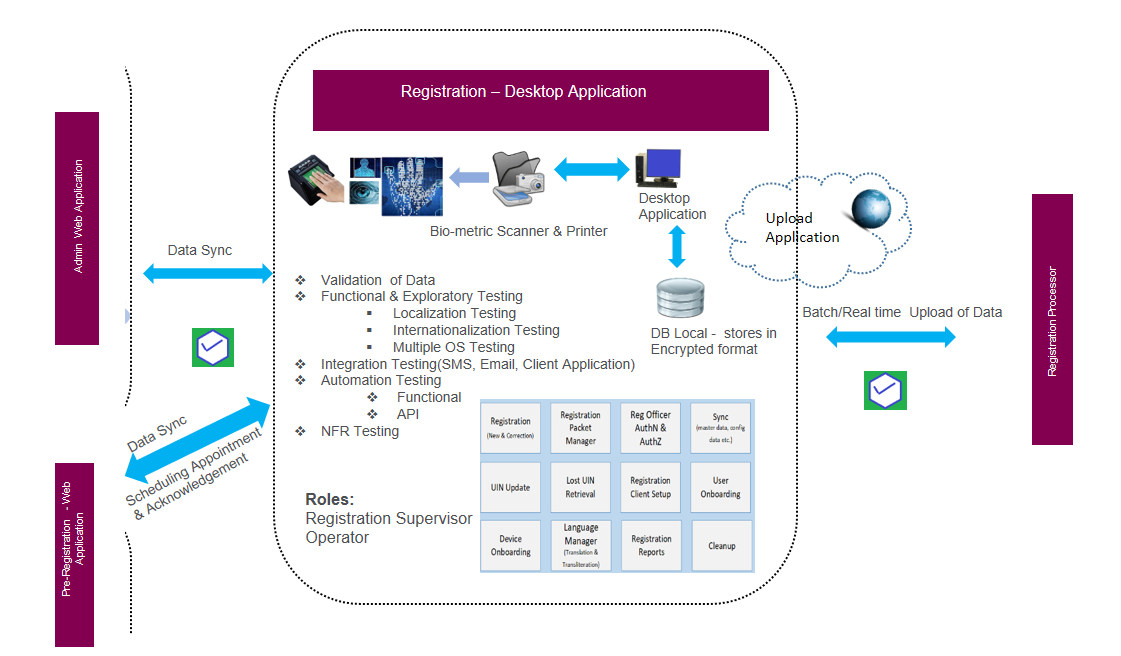


Figure 7 : Registration client Testing

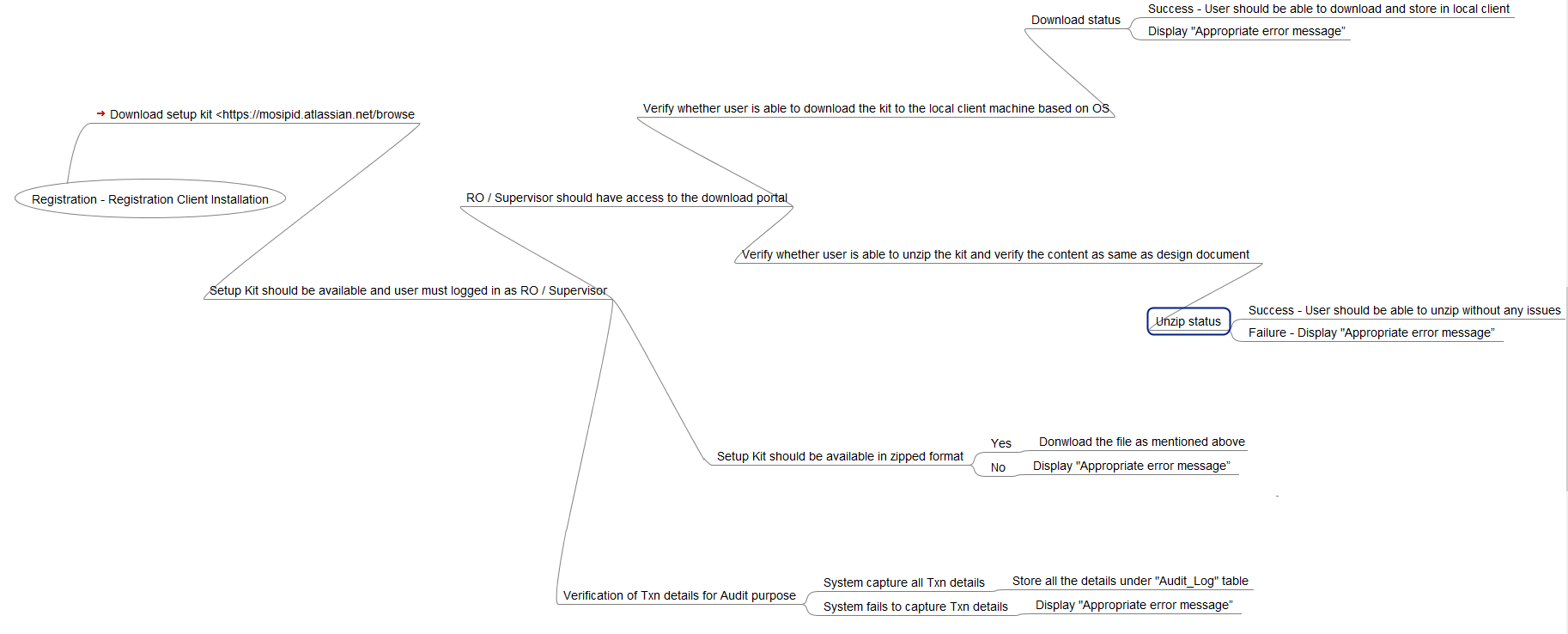
### Functionalities to test for Registration Client

* Forced Capture
* UIN Update
* Lost UIN
* Biometrics Capture: Iris, Finger Prints, Photo
* Biometrics Quality Check > Segmentation
* Translation / Transliteration
* Opt to Enrol - Demographic Data Capture
* Document Upload
* Registration by Category: INF/CHD, Regular Category
* Download Pre-registration data, Secure Packet Transfer to client
* Sync Master Data-Auto vs User – Client to server packet notification
* Secure Biometric Login (Operator Authentication) / Logout – Session timeout
* Virtual Machine – Dongle Bootable
* Register EC against server
* On boarding devices, device registration
* Operator Authentication – Biometric, Multi-factor
* Secure Packet Storage / Encryption – Using public key of server
* Acknowledgement Receipt – View/Gen. pdf (Registration no. gen)
* Notification: Email/SMS
* Upload Registration: Correction
* Audit Logging – Capture Usage Metadata, Security Audit & Scanning
* Client Manageability: Server Auto Update, Server Patch Upgrades
* Client Security: Secure Data from Theft, Tampering, Virus/Malware, Periodic scans
* Packet Upload
* Packet upload from DMZ to server, Ack. on registration decryption, Packet clean-up

### Test Approach in validating Registration Client

### Functional verification

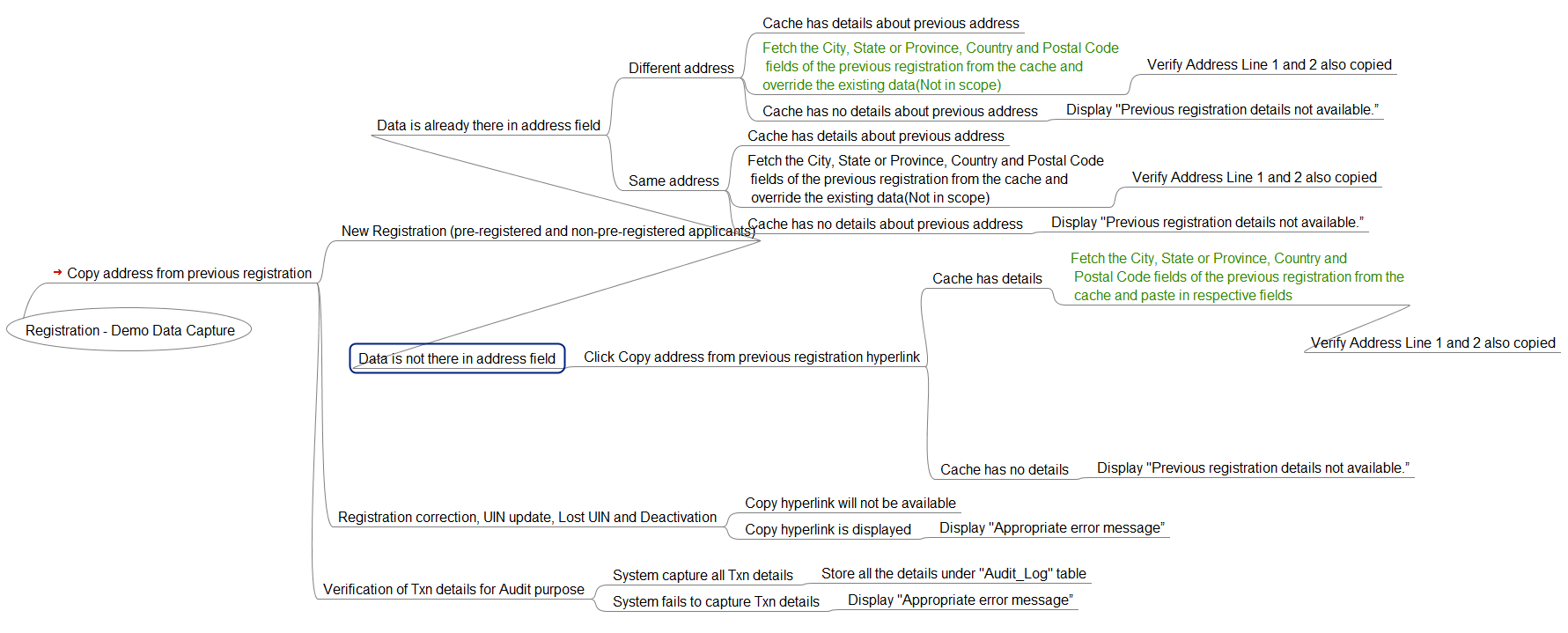
* + Registration Client Installation



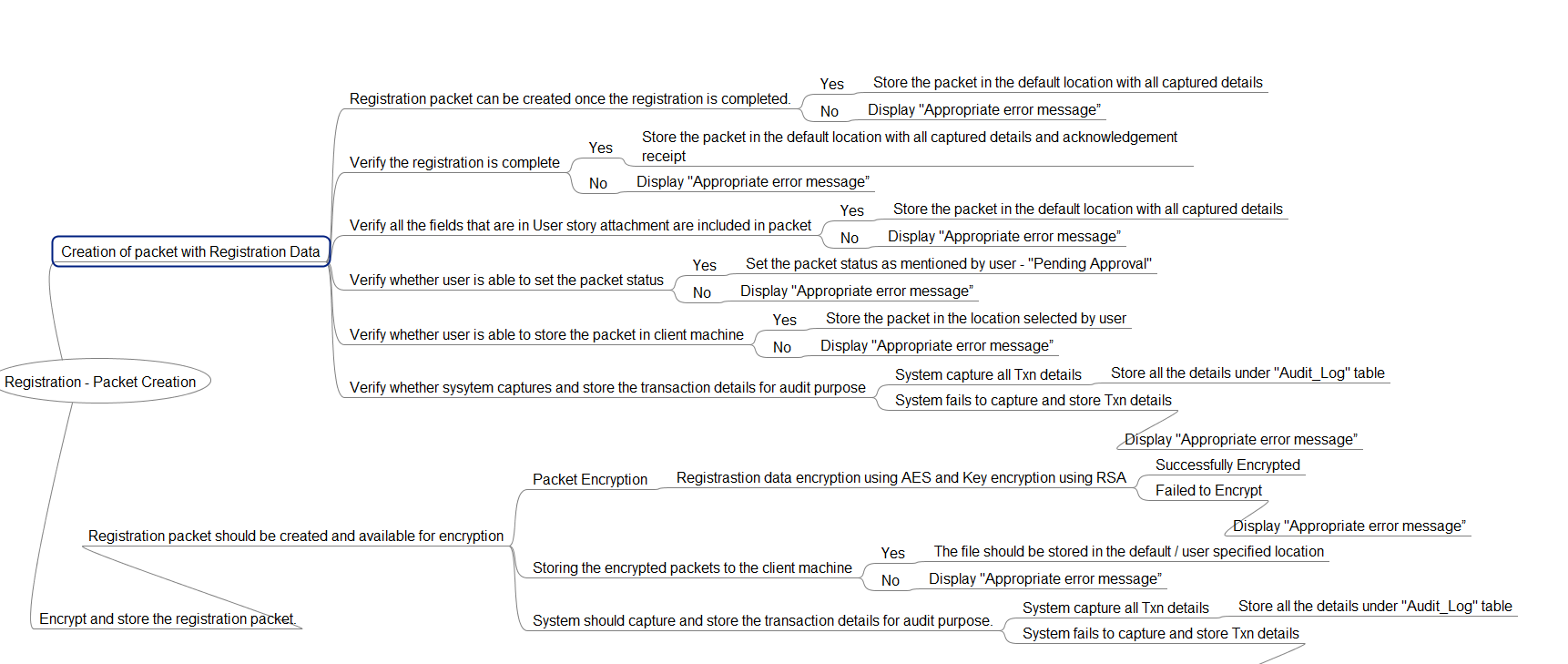
* + Registration Client Launch –



* Registration Demographic Data Capture –



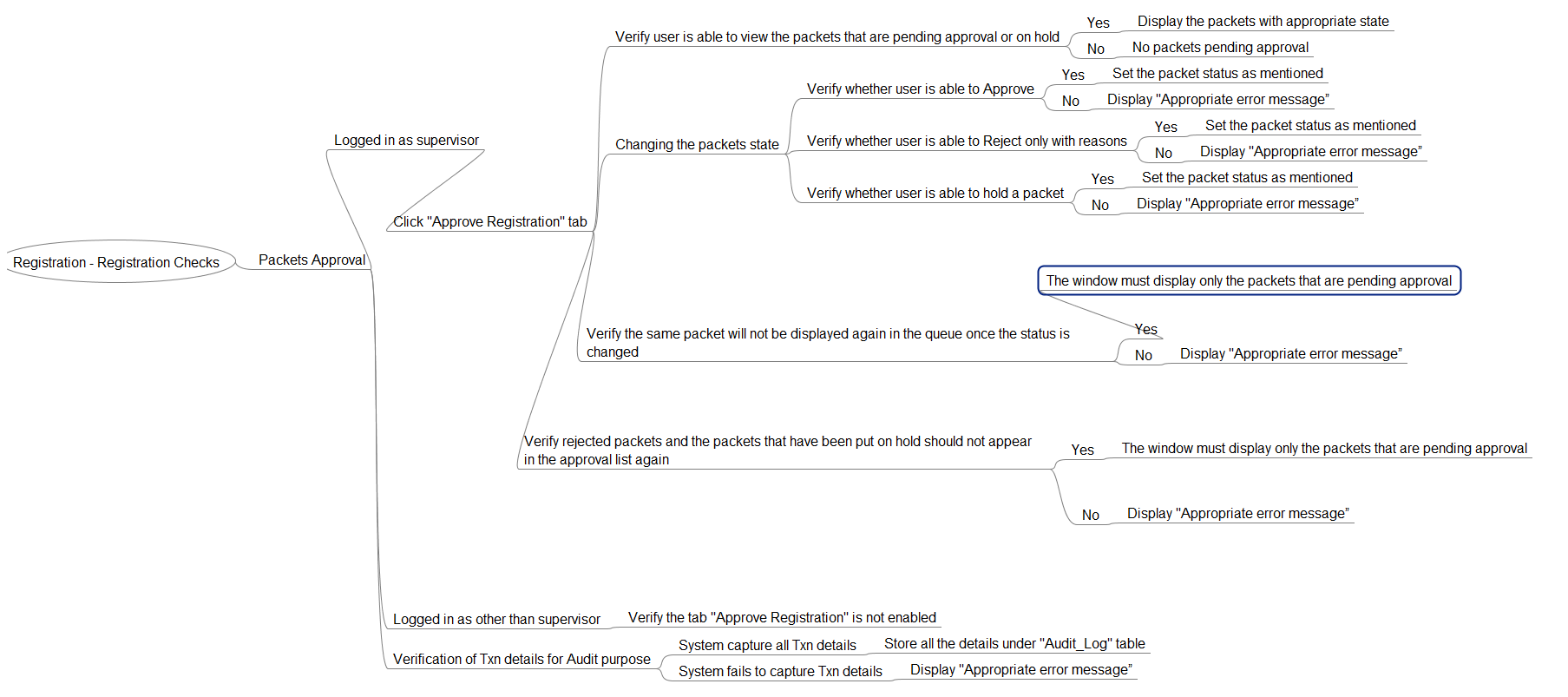
* Registration Packet Creation –



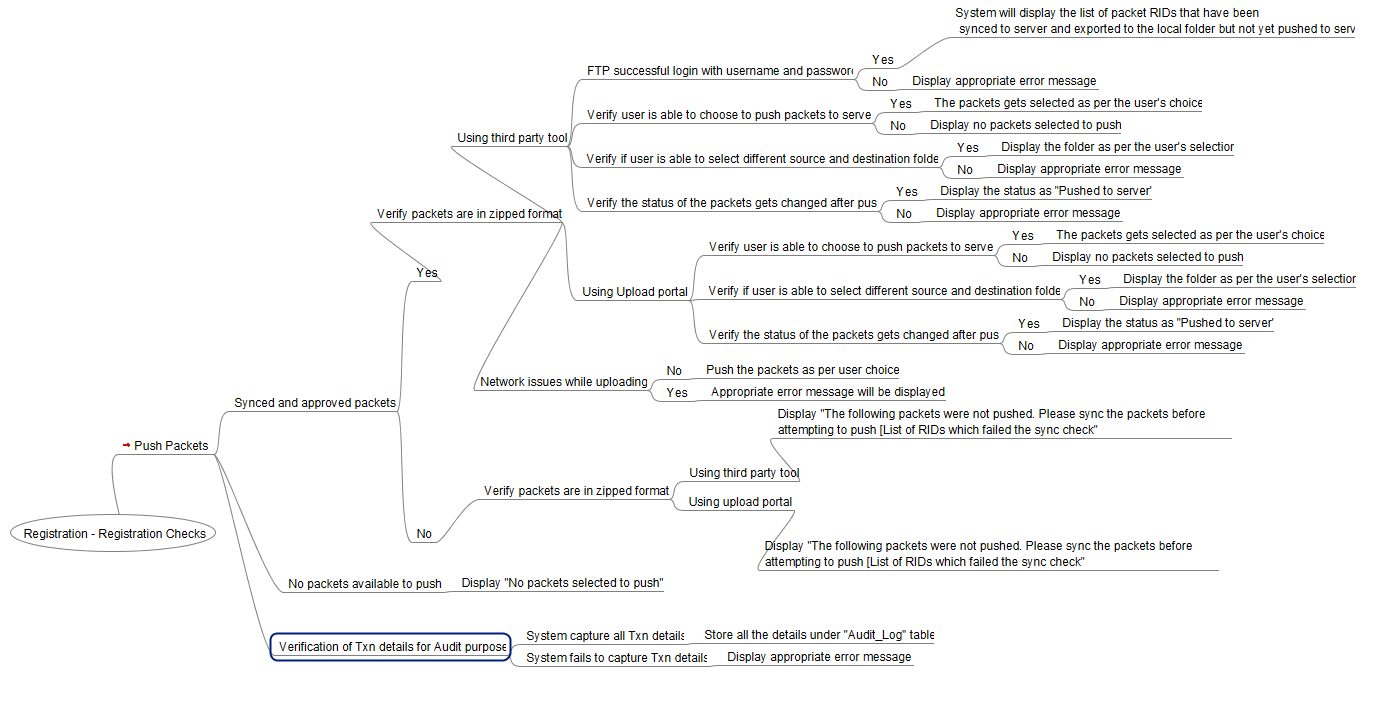
* Registration Acknowledgement –



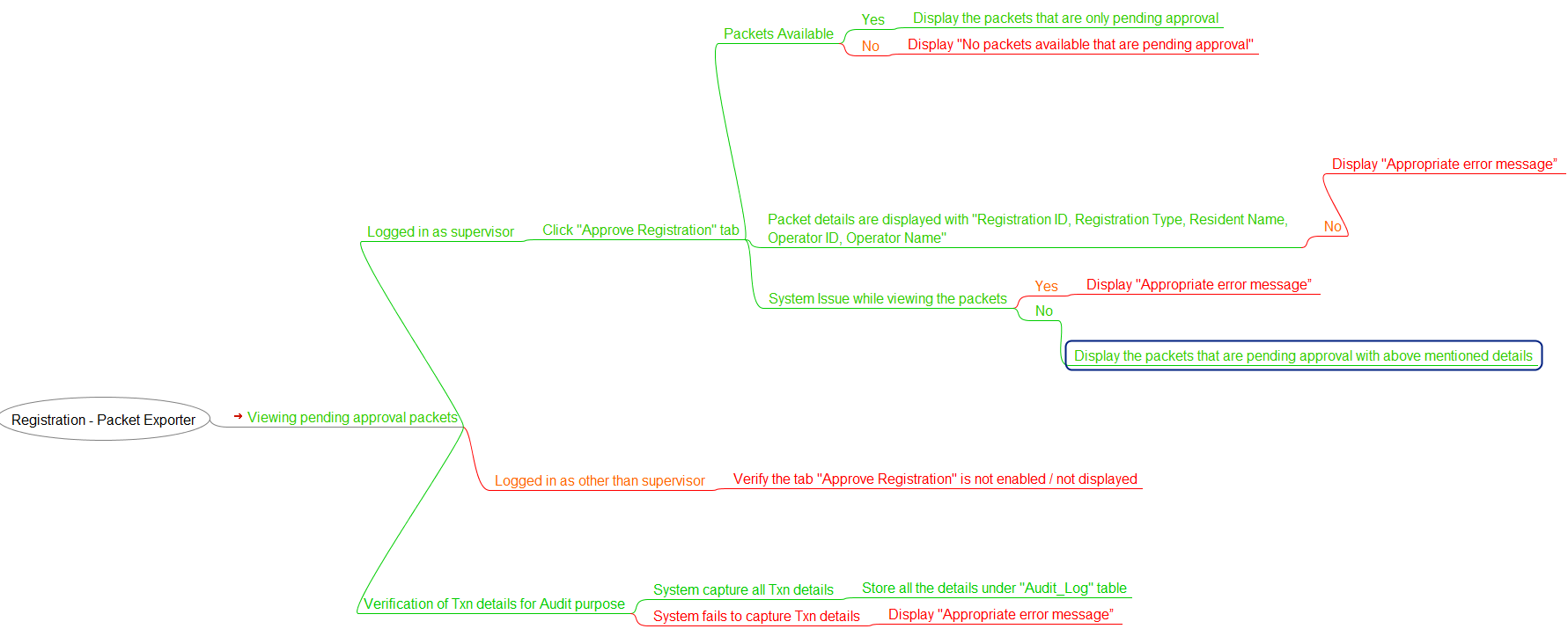
* Registration Packet Approval –



* Registration Packet Processing–

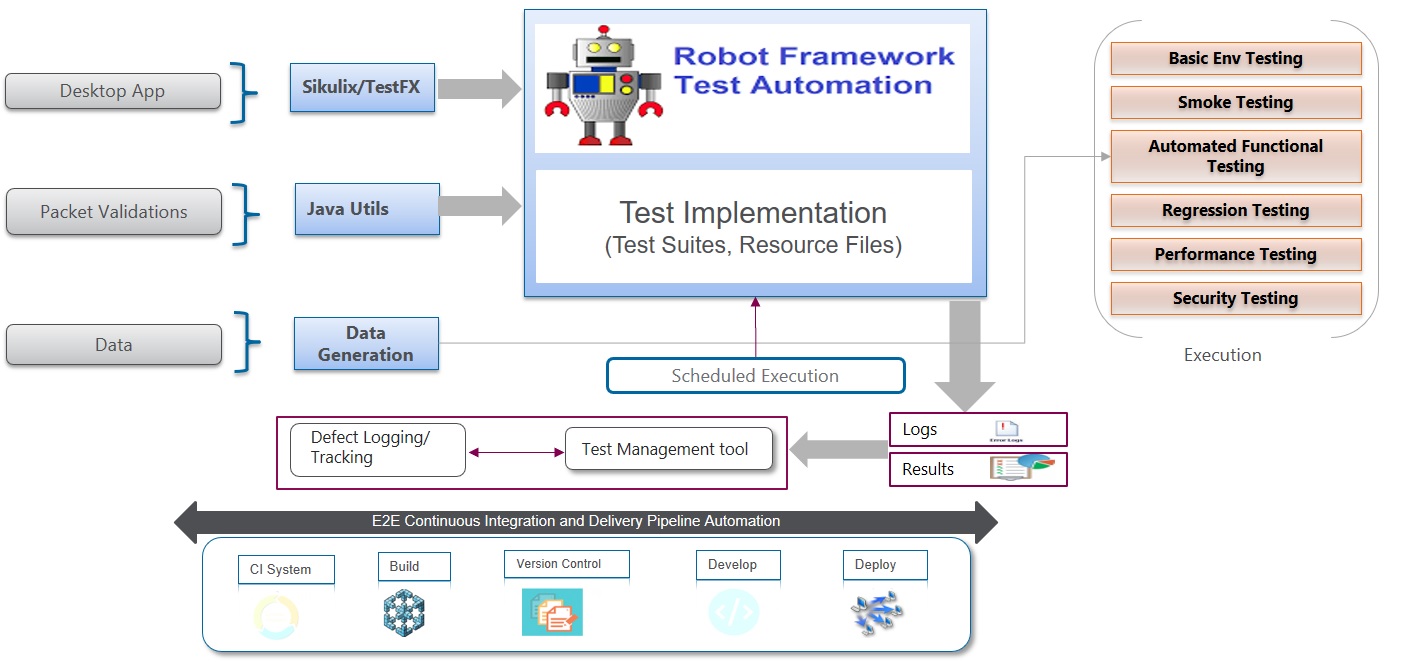


* Registration Packet Exporter –



### Automation Testing – Registration Client

### Functional Automation – Desktop app



Details of testing for Functional thick client Automation

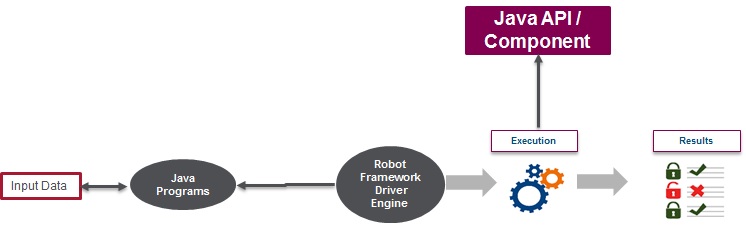
|  |  |
| --- | --- |
| Objective | Functional automation testing is performed to verify that the defined functionalities / requirements of the Registration client applications are met and the functions conform to the design & architecture specifications in an automated fashion.  Some potential scenarios where functional automation testing can be considered are:   * Synchronize pre-registration data * Edit demographic data with transliteration * Identification document capture (*content validation will be manually validated)* * Biometric data like finger print, iris and photo capture (*Pre-defined images in the format of json/xml file will be considered as input*). Biometric data capture through devices will be manually validated * Transfer captured data into centralized server (Registration Processor) (*This will be manually validated, although scripts would be developed for faster execution*) |
| Preparation Activities | This is an integral part from functional validation perspective which would be the building block considering the complete application. Listed below are the preparation activities which are expected to be performed:   1. Understanding the functionality from an individual module / application perspective as well as from an integrated mode considering the holistic approach 2. Understanding the functional and technical requirements and specification documents 3. Understand functionalities that can be automated 4. Identification of automation test scenarios based on the defined specification document and use cases 5. Perform ROI analysis on the same 6. Walk through the test scenarios with the stakeholders for alignment. 7. Creation of Test Data need for execution 8. We shall be using <http://dsl.cds.iisc.ac.in/projects/Anguli/> for usage of biometric data to test the Registration Module 9. Reading the external file /JSON format which stores the pre image scanned of users mapped with a Photo, 10 fingers and 2 eyes with a unique identifiers that may/shall be invoked from the script as the Test Data for the validation of the scenario. 10. Creation of mock service virtual scripts for components unavailable (i.e. Pass through the Bar code scan/ QR scan code validation, Creation of the acknowledgement for registration, passing the OTP into the system to proceed through the authentication, Once the document is uploaded to create a positive or negative ACK) 11. Creation of automation test scripts based on the scenarios and then prioritized based on the criticality 12. Create Requirement traceability matrix (RTM) from specification document 🡪 Use case 🡪 Test scenario 🡪 Test cases (we shall be using ConformIQ tool for the RTM) |
| Execution Approach | The automation tool going to be used will either be TestFX or Sikuli library to automate the Java based Desktop Registration Application using Robotframework.   1. **Sample Test Definition/Scenario** : Registration Officer log-in to client application through UID and biometric login   **Automation Validation Points** :  Pre-defined Test Data will be created for a set of registration officer along with UID.  The Automated scripts will validate the biometric authentication of the registration officer along with the UID and allow him/her to proceed to login to the system.  pre-registrationdata is loaded into their system   1. **Sample Test Definition/Scenario** : Resident visits the registration Centre with a pre-registrationacknowledgement number   **Automation Validation Points** :  The Automated scripts to enter the pre-registrationacknowledgement number and retrieve all the basic information provided/available in the Database.  The Data will be mocked, the verification of the texts in the pre-registrationwill be fully validated during integration phase only   1. **Sample Test Definition/Scenario** : The Registration officer would facilitate capture of    * 1. demographic data,      2. biometrics data,      3. audit events      4. encryption/decryption of collected data as packets      5. receipt of acknowledgement for collected packets   **Automation Validation Points** :  The Automated scripts to validate different type/format of the docs allowed as per the config file and seek for a successful ACK from the Mock/API server.  A Mock script would be created to generate an ACK from Database post the upload.  A Mock/stub will be created to generate the acknowledgement for Registration and validate the reporting as per the specification  The following activities are carried out as part of this activity:   1. Post all the preconditions are met the application under test will be validated against the test scripts defined based on the specification documents. This execution takes the precedence based on the priority. 2. The scenarios would be tested on Windows OS. For Linux OS manual validation would be done as the reusability of test scriptsto be executed Linux OS may not be feasible as this open source tool is based on Image based recognition. 3. The primary execution will be planned for the Windows OS 4. Defect cycles are initiated based on the failed cases as well the bugs detected during the execution. 5. Expectations on the exit criteria are evaluated towards sign off to take call on go – no go decision on the functionality to next phase. |
| Prerequisites | 1. Test environment is available with appropriate data 2. Necessary test data is created for the testing to be executed 3. Mock service scripts are ready and available 4. Test Strategy, Plan & Scenarios are signed off |
| Environment | Test environment |
| Responsibility | Test team (automation testing associates) |
| Other Considerations | The above solution is based on the current understanding of the MOSIP landscape |

### Testing of API libraries

The Registration module would comprise of the following API services

|  |  |
| --- | --- |
| LoginService | RegistrationApprovalService |
| MapMachineService | RegPacketStatusService |
| PacketHandlerService | SyncStatusValidatorService |
| PacketUploadService | TemplateService |

For test validation of each of the Java API a custom Java program will be developed and will be executed through the Robotframework test suite as depicted below



### Performance Testing – To be Updated

## Registration Processor

### Objective

REGISTRATION PROCESSOR is the core part of MOSIP where REGISTRATION PROCESSOR module will be responsible for processing of registration packets through the phases of acceptance, validation, processing and dispatch of UIN.

### Main features of Registration Processor are

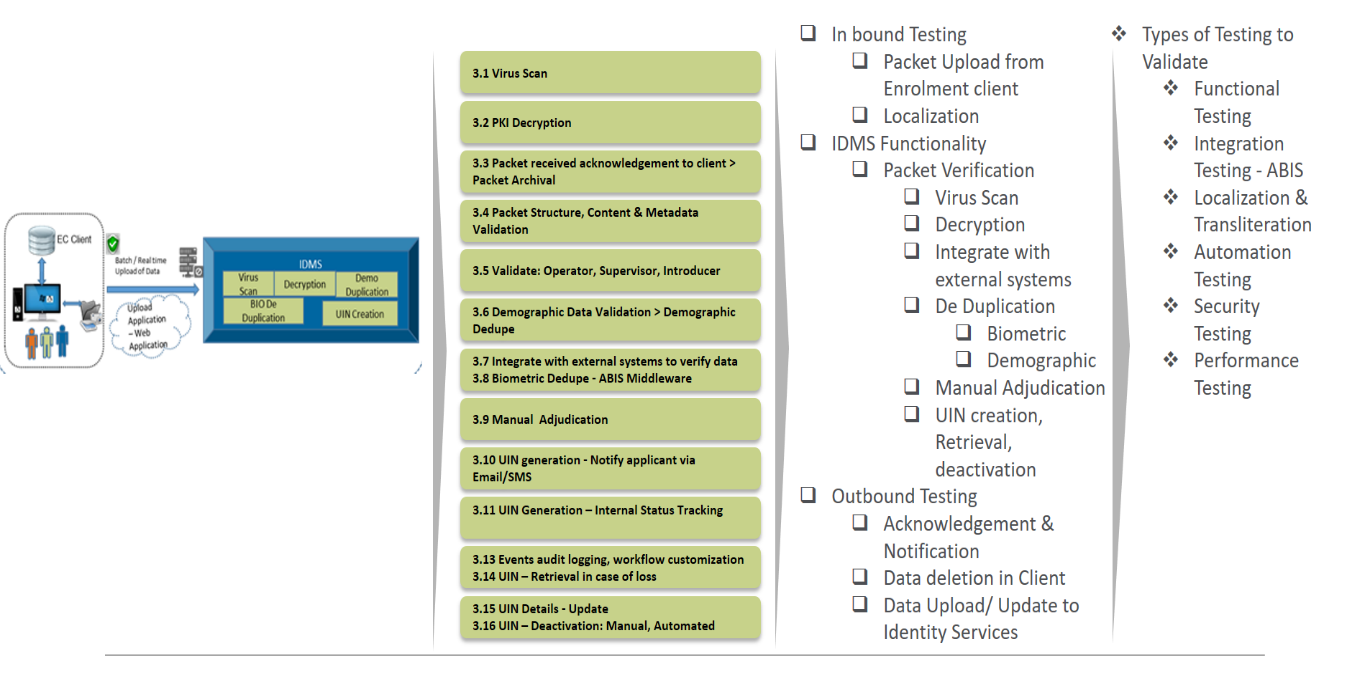


Figure 10 : REGISTRATION PROCESSOR Testing

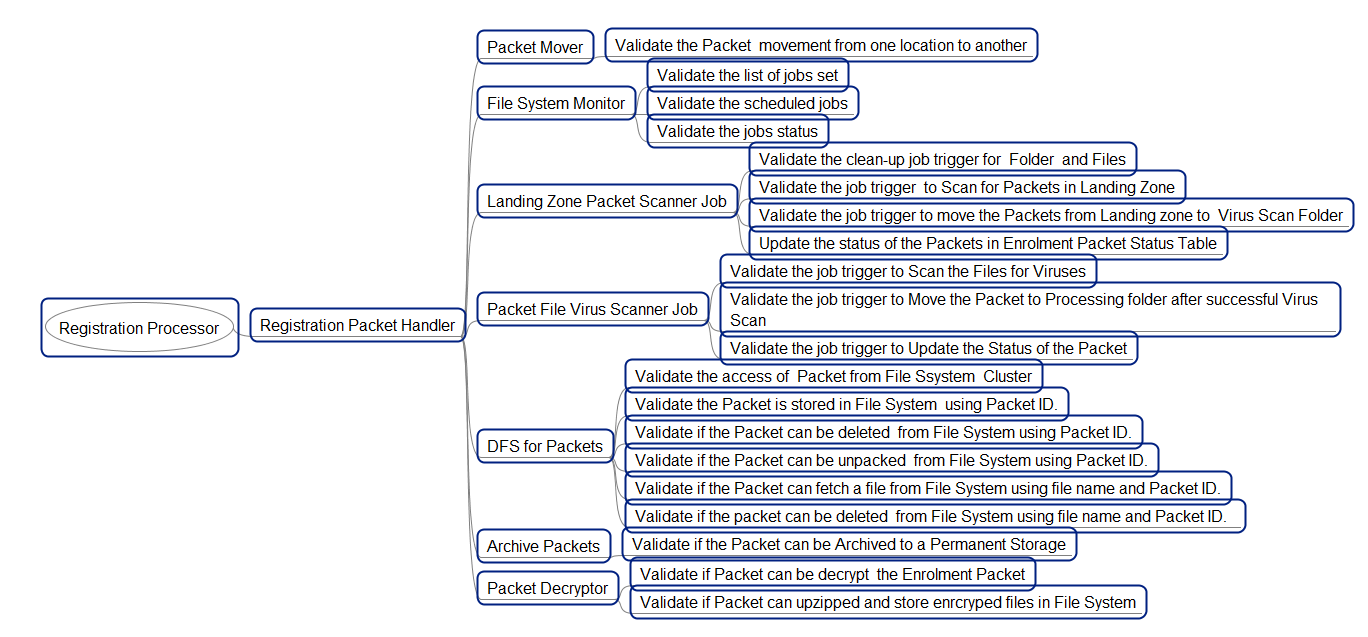
### Functionalities to test for Registration Processor

* + Virus Scan
  + PKI Decryption
  + Packet received acknowledgement to client > Packet Archival
  + Packet Structure, Content & Metadata Validation
  + Validate: Operator, Supervisor, Introducer
  + Demographic Data Validation > Demographic Dedupe
  + Integrate with external systems to verify data
  + Biometric Dedupe - ABIS Middleware
  + Manual Adjudication
  + UIN generation - Notify applicant via Email/SMS
  + UIN Generation – Internal Status Tracking
  + Events audit logging, workflow customization
  + UIN – Retrieval in case of loss
  + UIN Details - Update
  + UIN – Deactivation: Manual, Automated

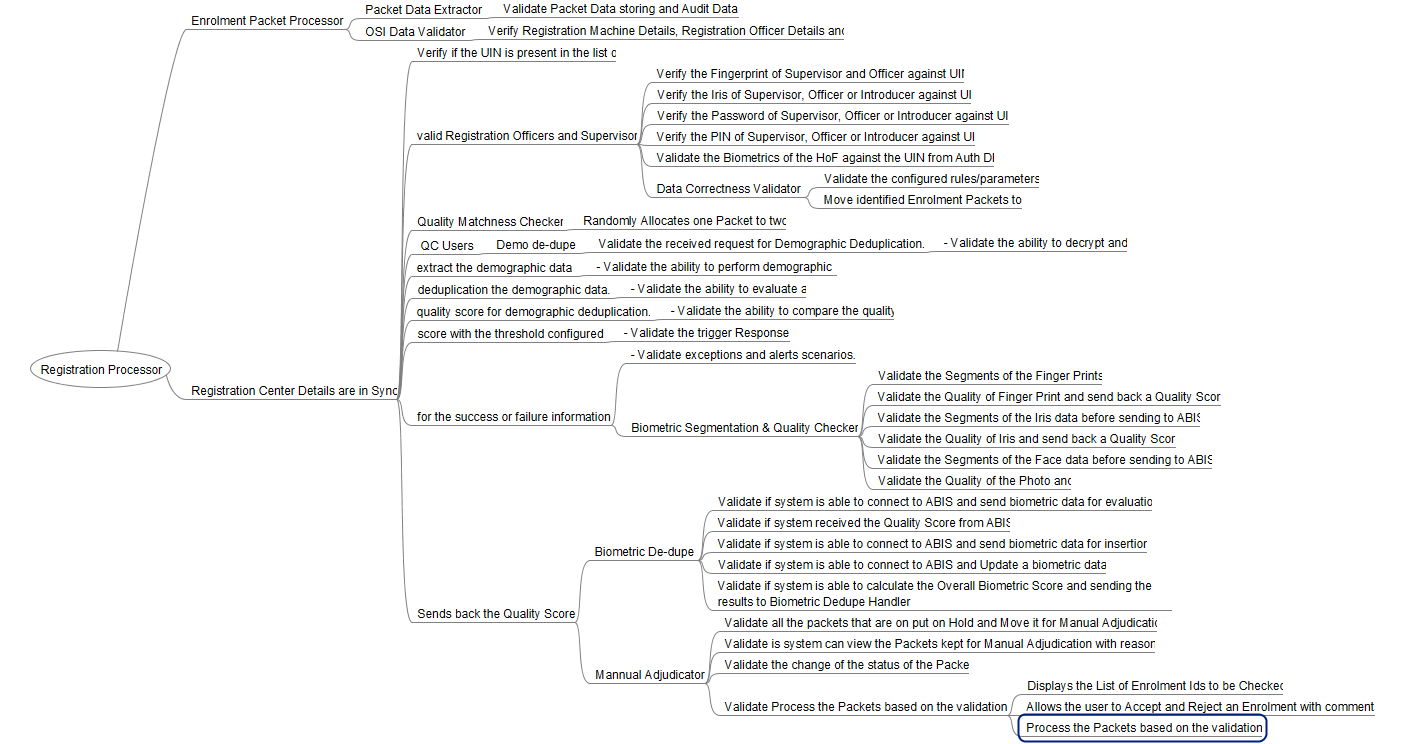
### Test Approach to validate Registration Processor

### Functional verification

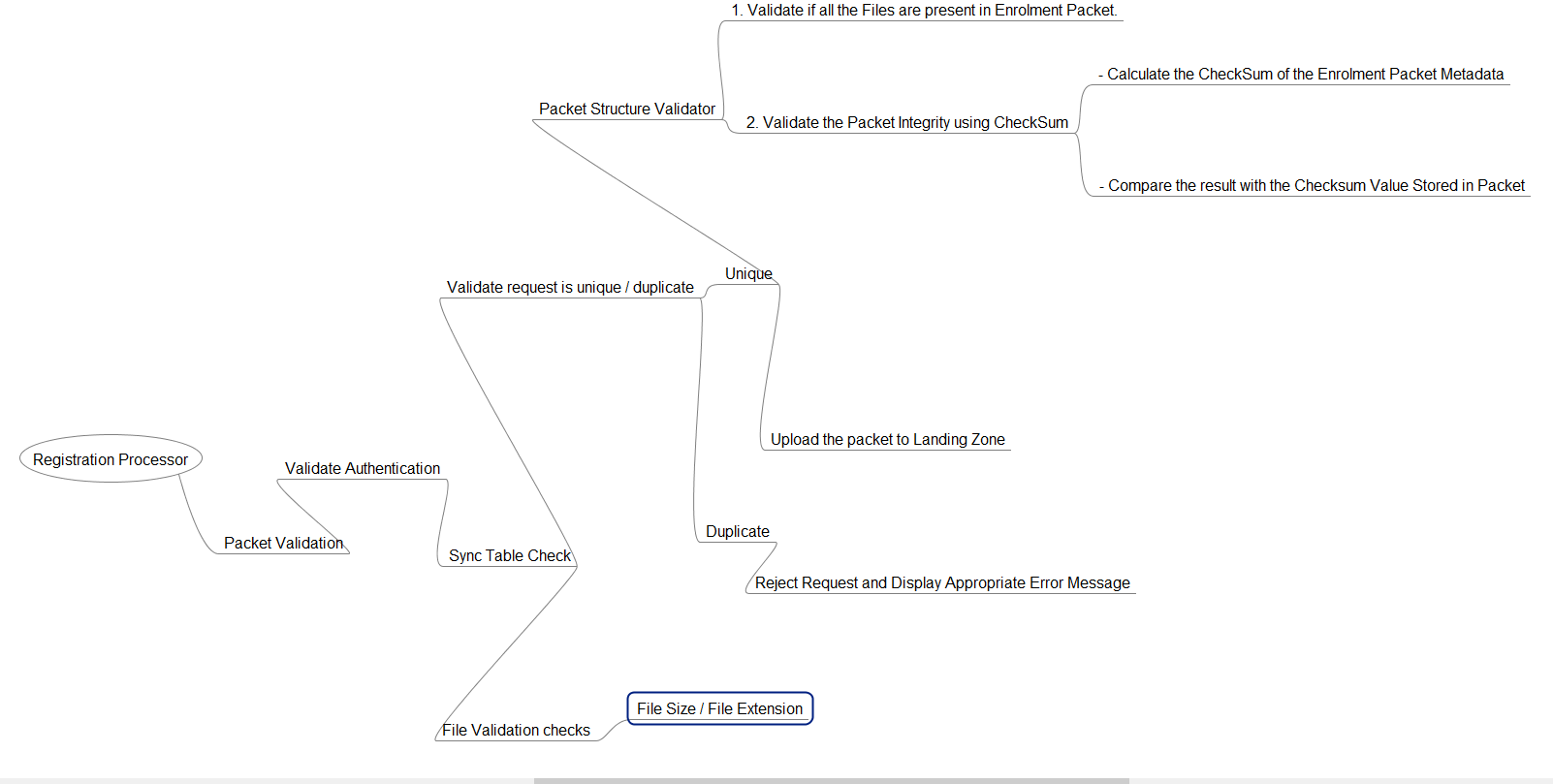
* Registration Processor Packet Handler –



* Registration Processor Packet Extractor –



* Registration Processor Packet Validation –



* Registration Processor Minutiae Extractor –

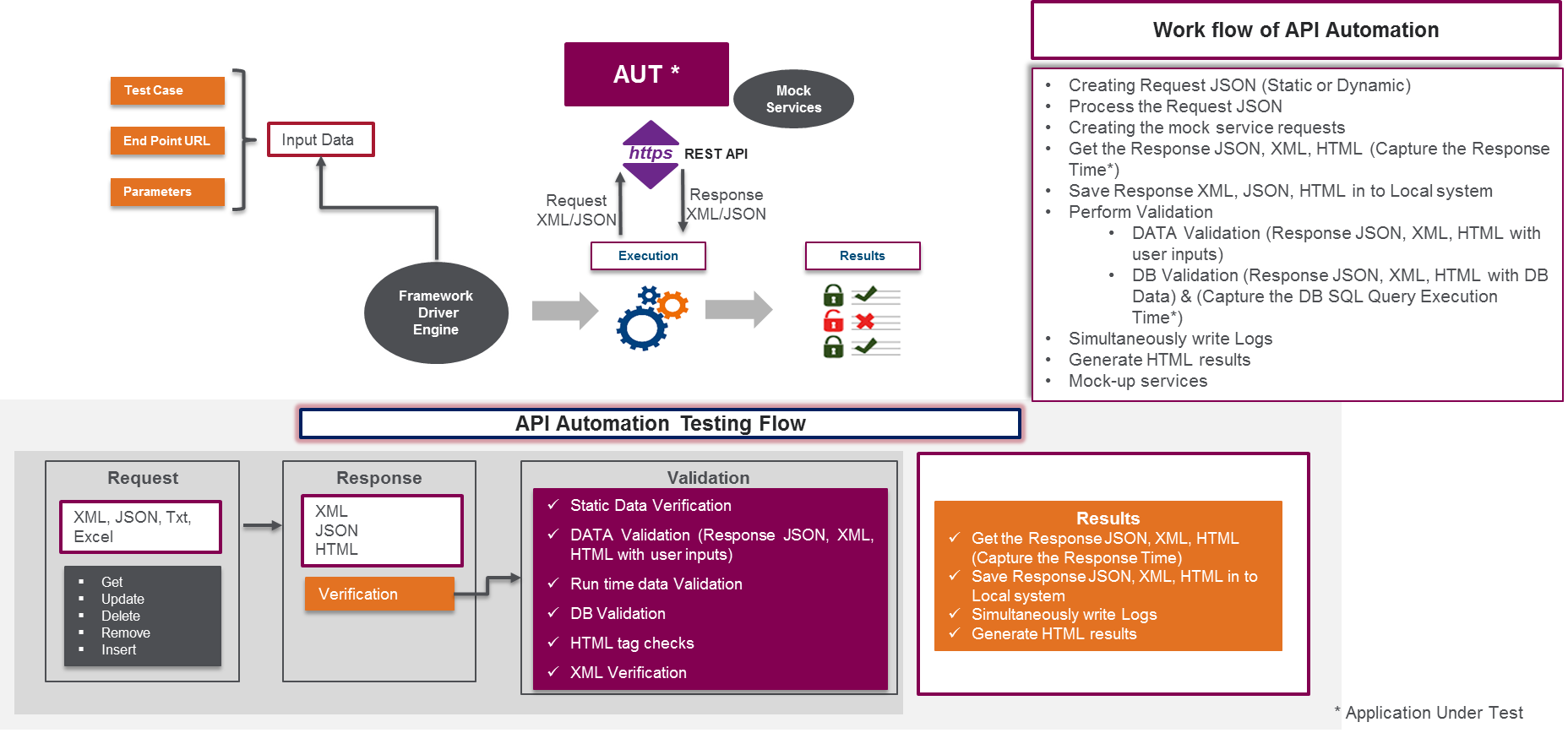


### Automation Strategy for Registration Processor

###### API / Micro Services Automation

Following Restful Web services would be tested:

|  |  |
| --- | --- |
| saveOrUpdate(UserMachineMappingDTO0 | getBlockedUserCheck(userId) |
| view() | getOTP(key) |
| validateUserPassword(userId, hashPassword) | validateOTP(key) |
| getUserDetail(userId) | handle(RegistrationDTO) |
| getCenterName(centerId) | pushPacket(Packet) |
| getRegistrationCenterDetails(centerId) | updateStatus (uploadedPackets List) |
| getRoles(userId) | getAllEnrollments() |
| createReceipt() | packetUpdateStatus(id,clientStatusCode, approverUserId,statusComments,updBy) |
| packetSyncStatus() | validateSyncStatus() |



Details of testing for API/Micro services Automation

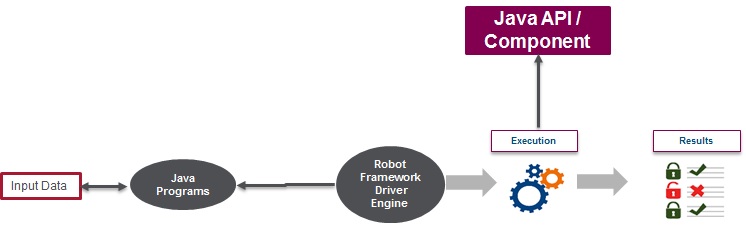
|  |  |
| --- | --- |
| Objective | The objective is to test the business layer logic of the software of   1. Java API services   which is been implemented using REST protocol to carry JSON format data over HTTP endpoint |
| Preparation Activities | API /Web Services Testing will basically involve   * Determining the operations the web service provides * Determining the XML/JSON request format which the tester needs to send * Determining the response XML/JSON format * Using a tool to send request and validate the response * Will be using mock service request to fill the gap of missing components for requests/response   Web Service Test Automation library like Karate or Rest-assured will be used in this project which will be integrated within the Automation framework.  Mock services will be created either using Wiremock or Citrusframework for completing the validation flow in absence of the DB or the related API component |
| Execution Approach | Entire web services testing will be automation. The automation framework will be comprising of Karate tool and Rest-Assured jars for complete testing of the micro services and the API.  For each of the API   1. Understand the functionality of the API program and clearly define the scope of the testing activity 2. Apply testing techniques such as equivalence classes, boundary value analysis and error guessing and write test cases for the API 3. Input Parameters for the API need to be planned and defined appropriately 4. Execute the test cases and compare expected and actual results |
| Prerequisites | 1. Components, Database and server if not available will be attempted to pass-through using a mock script 2. Parameter selection and categorization required to be known to the testers 3. All communication & data exchange will be based on XML/JSON 4. Assumption that there is no need of virtualization environment to be created |
| Environment | Test environment |
| Responsibility | Test team (automation testing associates) |

#### Testing of Java API libraries

The Registration Processor module would comprise of the following Java API services

|  |  |
| --- | --- |
| PacketReceiverService | FileManager |
| RegistrationStatusService | PacketInfoManager |
| SyncRegistrationService | EventBusManager Description : Declares all the methods to be used by Processor stages |
| TransactionService | FileSystemAdapter Description : This is an adapter interface to communicate with distributed packet store. |

For test validation of each of the Java API a custom Java program will be developed and will be executed through the Robot framework test suite as depicted below



### Performance Testing – To be updated

## Identity Services / Authentication - IDA

### Objective

The identity services application validated against the Authentication and integration with Trusted Service Providers in the frontend. The validation of application functionality by the biometric matching from the authenticity of resident are tested (who asked to authenticate by a user agency)

### Main features of IDA

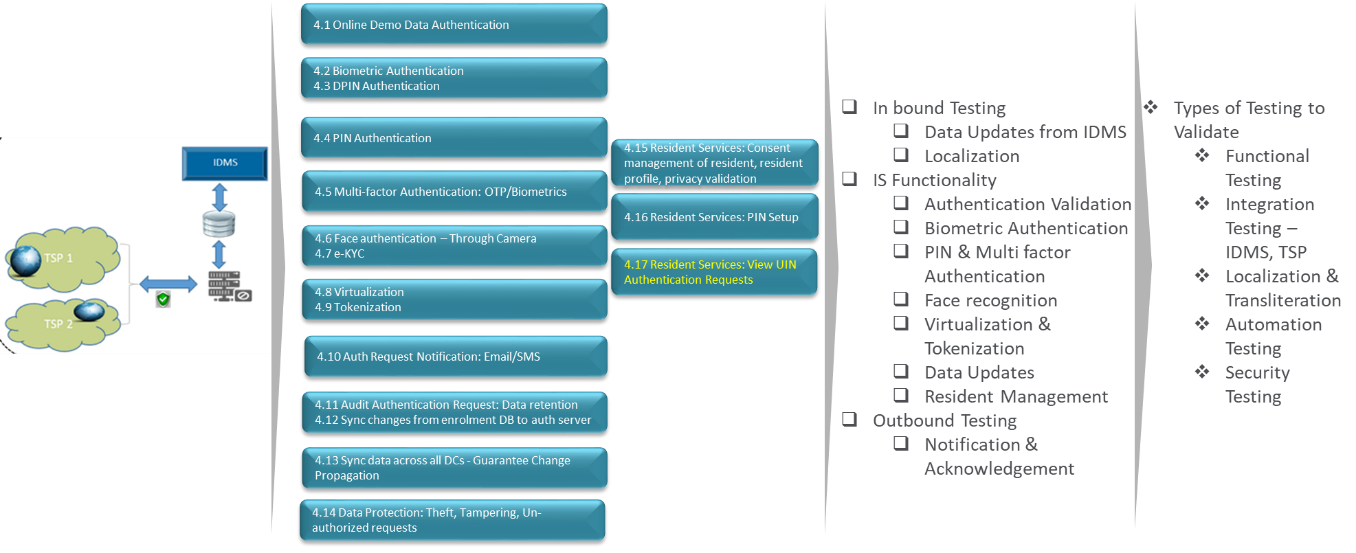


Figure 12 : Identity Service Testing

### Functionalities to test for IDA

* Online Demographic data authentication
* Biometric Authentication
* DPIN Authentication
* Multifactor Authentication (OTP/Bio Metric)
* Face Authentication
* E-KYC
* Virtualization & Tokenization
* Audit Authentication Request
* Authentication request notifications
* Audit Authentication Request: Data retention
* Sync changes from registration DB to Authentication server
* Sync data across all DCs - Guarantee Change Propagation
* Data Protection: Theft, Tampering, Un-authorized requests
* Resident Services: Consent management of resident, resident profile, privacy validation
* Resident Services: PIN Setup
* Resident Services: View UIN Authentication Requests

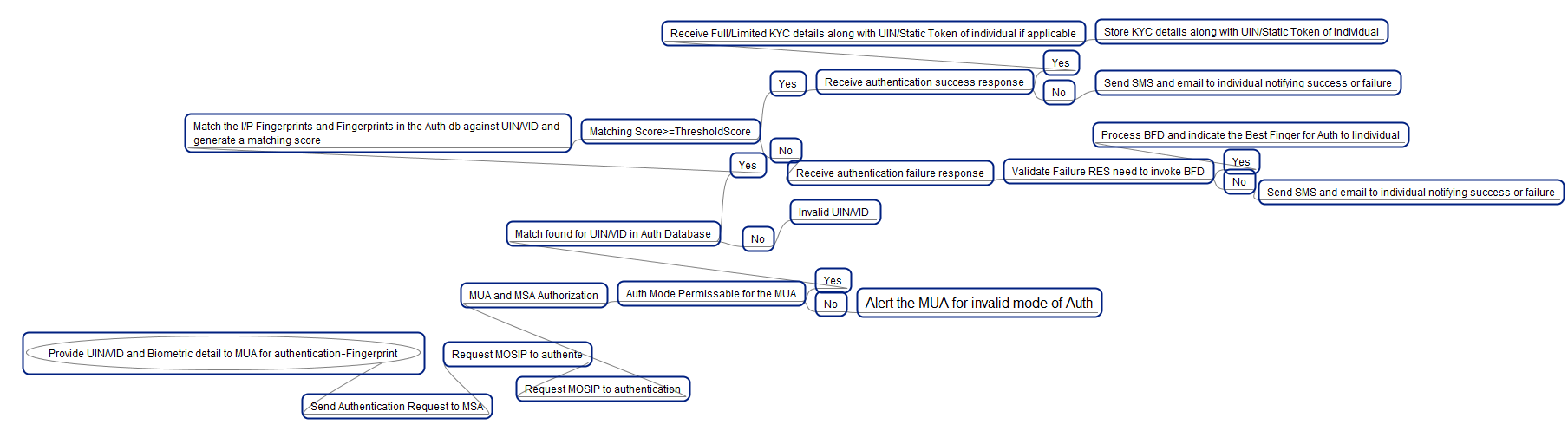
### Test Approach in validating IDA

### Functional verification

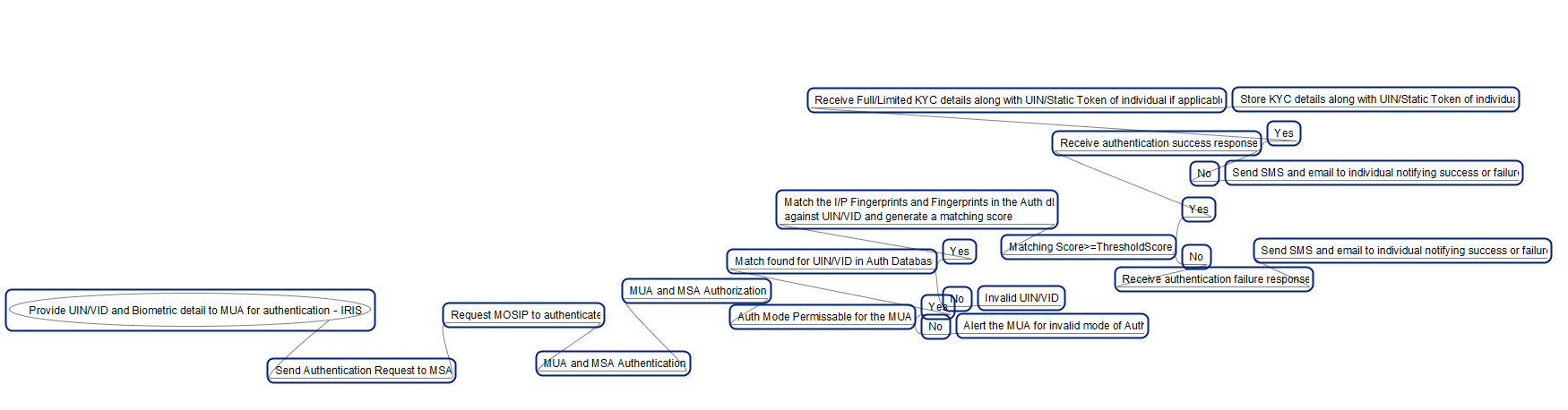
* IDA- Demo Authentication –

###### 

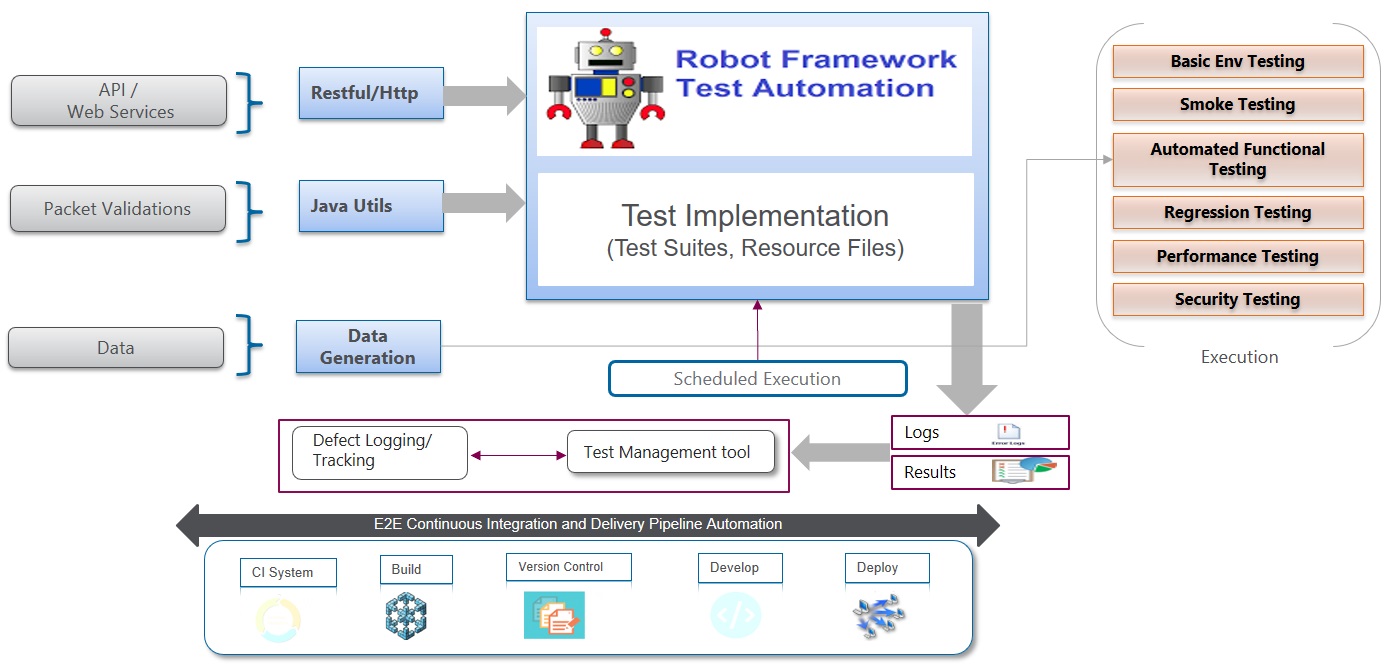
* IDA- Fingerprints Authentication –



* IDA- IRIS Authentication –



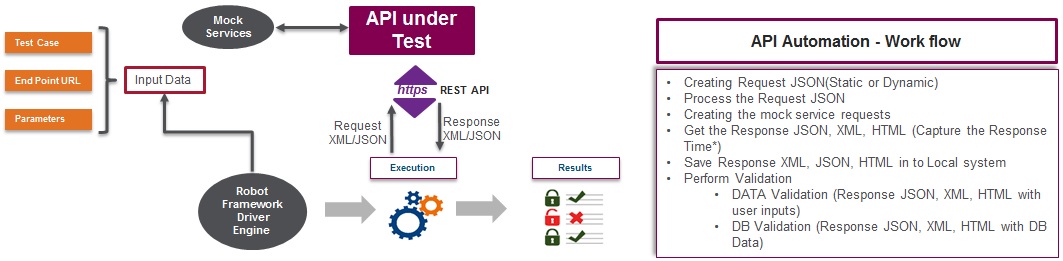
#### The approach for automation for the ID – Authentication module will be as follows



###### API / Micro Services Automation

Following Restful Web services available in the ID-Authentication module will be tested and automated for contract testing, functional testing, performance testing and security testing. Necessary stubs or mock services will be created for the upward/downward application as appropriate to continue with the testing

|  |  |
| --- | --- |
| OTP Trigger | Iris Auth |
| OTP Auth | Face Auth |
| Demo Auth | Multifactor Auth |
| Static Pin Auth | e-KYC Auth |
| Fingerprint Auth | Best Finger Detection |

****

###### Details of testing for API/Micro services Automation

|  |  |
| --- | --- |
| Objective | The objective is to test the business layer logic of the software of   1. Java API services   which is been implemented using REST protocol to carry JSON format data over HTTP endpoint |
| Preparation Activities | API /Web Services Testing will basically involve   * Determining the operations the web service provides * Determining the XML/JSON request format which the tester needs to send * Determining the response XML/JSON format * Using a tool to send request and validate the response * Will be using mock service request to fill the gap of missing components for requests/response   Web Service Test Automation library like Rest Instance or Rest-assured will be used in this project which will be integrated within the Robotframework.  Mock services will be created either using Wiremock or Citrusframework for completing the validation flow in absence of the DB or the related API component |
| Execution Approach | Entire web services testing will be automated. The automation framework will be comprising of Rest-Instance or Rest-Assured jars for complete testing of the micro services and the API.  For each of the API   1. Understand the functionality of the API program and clearly define the scope of the testing activity 2. Apply testing techniques such as equivalence classes, boundary value analysis and error guessing and write test cases for the API 3. Input Parameters for the API need to be planned and defined appropriately 4. Execute the test cases and compare expected and actual results |
| Prerequisites | 1. Components, Database and server if not available will be attempted to pass-through using a mock script 2. Parameter selection and categorization required to be known to the testers 3. All communication & data exchange will be based on XML/JSON 4. Assumption that there is no need of virtualization environment to be created |
| Environment | Test environment |
| Responsibility | Test team (automation testing associates) |

### Performance Testing

|  |  |
| --- | --- |
| Scope | **In Scope**   * Testing for MOSIP authentication services - Web Services / Micro services Performance Testing   **Out of Scope**   * Performance testing of 3rd party Automated Biometric Identification System (ABIS) interfaces |
| Objective | The objectives of performance testing of authentication services are   * To determine **number of authentication requests processed per day (8 hours)** and also ensuring that there are no bottlenecks in the system. * To determine if Authentication services application can scale if additional hardware is provided - (TBD - vertical / horizontal) * To determine if Authentication Services application can withstand continuous usage for 12 / 24  hours –   Note: Actual numbers for each of the objective will be decided later in consultation with architects and business analysts |
| Preparation Activities | Listed below are the preparation activities which are expected to be carried out:   1. Understanding the key application functionality and understanding critical features of the application 2. Understanding the functional and technical requirements and specification documents. Use cases can be employed to provide clarity of the desired functionality 3. Identification of services based on the defined specification document and use cases    * Frequently used web services / micro services    * Business critical web services / micro services 4. Walkthrough of the web / micro services with the stakeholders for alignment 5. Create workload models for identified services / workflows 6. Prepare test plan including scope, workload model and SLAs / KPIs for performance 7. Prepare seed test data and pump it into registration DB (instead of initiating testing with blank DB)   Mock services will be created either using *Wiremock* or *Citrusframework* for completing the validation of performance in absence of the DB or the related API component  Note: we will reuse mock services created for test automation purpose |
| Execution Approach | **Preparation of seed test data into Registration DB**  We will add sufficient number of records in the Registration DB as seed data before we commence actual performance testing of Authentication Services application.  Performance testing will focus on understanding server capacity to handle load in terms of requests to process registration packets.  Keeping in mind objectives for MOSIP we will carry out following types of testing:   1. **Load Test:** This involves testing the application under normal expected load by way of gradually increasing the load. Web services / micro services will be tested for gradually increasing number of requests.   For example, 0.10x  0.25x  0.50x  1x of expected load  We will create different workload model to simulate real life scenarios.   1. **Scalability Testing:** This involves carrying out load tests by increasing number of requests to understand scalability of application with additional hardware. Purpose of scalability testing is to identify major workloads and mitigate bottlenecks that can impede the scalability of the application to meet the load demands of the coming years, in a graceful manner.   For example – 1.25x load  1.75x load  2x load with additional hardware   1. **Endurance Testing –** This involves testing services with sustained load over extended period of time. It helps to find the following kind of bugs:-    1. Resource leak    2. Timing    3. Hardware-related    4. Counter-overflow   For example – 1x load for 8 / 16 / 24 hours  **Server Monitoring**  Key parameters of underlying infrastructure would be monitored while carrying out above mentioned tests. |
| Prerequisites | 1. Application is functionally stable without any S1/S2 defects 2. Necessary test data is created for the testing to be executed 3. Test Strategy, Plan & Scenarios are signed off 4. Test environment is available with appropriate data 5. Appropriate access permissions are setup for server monitoring and for any other purpose identified in Test strategy / plan 6. Components, Database and server if not available will be attempted to pass-through using a mock script 7. Parameter selection and categorization required to be known to the testers 8. All communication & data exchange will be based on XML/JSON 9. Assumption that there is no need of virtualization environment to be created |
| Sample Scenarios | 1. 1 Trusted Service Provider 1 Language - 500 requests in 1 hour 2. 5 Trusted Service Provider – 1 Language – 2000 requests in 1 Hour 3. 25 Trusted Service Provider – 2 Languages – 5000 requests in 1 Hour 4. 25 Trusted Service Provider – 2 Languages – 50000 requests in 8 Hours |
| Environment | Test environment (preferably which mimics production environment closely) |
| Responsibility | Test team (performance testing associates) |

## Admin (Pre-registration, Registration and REGISTRATION PROCESSOR):

### Objective:

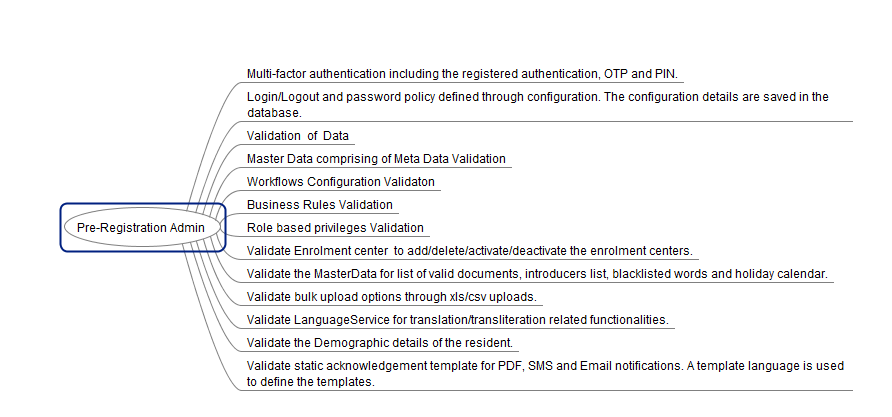
Admin functionality validated from the configuration and setup perspective where the impacted modules due to the configuration changes are tested.

Role based validation and the respective impact to the consuming application are tested.

Following are the functionalities to test from admin functionality perspective.

### Pre-Registration Client Admin:

* Login – Remember/Forgot/Change/Lock PW, Manage profile / Logout
* EC Config: Geotag, location vicinity, time availability, active workstations - xls/csv file upload
* EC Activation/De-activation
* Master Data Setup - Manual -Common Location Dictionary, List of valid docs, Introducers anagement, Blacklisted Words, Holiday Calendar
* Master Data Setup – xls/csv Common Location Dictionary, List of valid docs, Introducers management, Blacklisted Words
* Language/Labels Config for translation/transliteration
* Demo data field customization - Sequencing
* SMS/Email/PDF Template – Customization



### Registration Admin Functionalities:

* Configure mode of login to registration client
* Configure mode of operator authentication
* Blacklist operator/device data
* Client Security:ACL role management, authorize supervisory overrides/rules - xls/csv file upload
* Register Machines
* Configure the time span for locking the registration application if client offline
* Configure Finger Print Specs
* Configure threshold value: Finger Print, Iris
* Configure retry attempts for bio data capture

### REGISTRATION PROCESSOR Admin functionalities

* Configure parameters for batch process
* SMS/Email/PDF Template - Customization

## Security Test Strategy

The security objective is to design, recommend, and test security controls with defense-in-depth/layered security approach for protecting data in the MOSIP platform. The security controls and measures taken are considered in adherence to OWASP & SANS 25 best practices.

### Secure Code Review

Code review is one of the important processes in the secure SDLC. Code review process includes a combination of tool and manual technique to identify the security issues in the code. The main aim of the test would be to find all the vulnerable logics in the code. The approach is based on the information collected about the application in scope of the review.

Before we begin to review the code, it is always essential to identify the modules to be reviewed and determine the threats for each. The threats can be intentional or unintentional. Intentional threats are the piece of code where the anomalies are created or exploited on purpose eg. Backdoors. Unintentional threats arise from the piece of code where the developer has written code, which fails in a way which he has not anticipated.eg. Data validation bypass.

Once the threats are identified the next step would be to run the tool to detect/locate possible vulnerable points. The tool would check for various patterns in the source code that could possibly cause anomalies. The tool’s results are then analyzed for various risks.

Now a scenario based manual review of code is performed based on the results from the tool. A code walkthrough is performed for various scenarios understanding the complexity of the logic and possible anomalies. The threats could also possibly reside in the vulnerable points of the application. These vulnerable points are now reviewed manually, and a walkthrough is performed to identify any discrepancies.

Once all the threats are identified a report is prepared with all the identified threats and where possible and applicable proof of concept is attached for the identified vulnerabilities.

Below is the indicative list of test cases that will be performed during secure code review:

|  |  |
| --- | --- |
| **Areas** | **Test Cases** |
| **SQL injection** | Is the application susceptible to SQL injection?  Does the code use parameterized stored procedures?  Does the code use parameters in SQL statements?  Does the code attempt to filter input? |
| **Cross-site scripting** | Does the code echo user input or URL parameters back to a Web page?  Does the code persist user input or URL parameters in a data store that could later be displayed on a Web page? |
| **Data access** | Look for improper storage of database connection strings and proper use of authentication to the database. |
| **Input/data validation** | Does the code rely on client-side validation?  Is the code susceptible to canonicalization attacks?  Does the code validate data from all sources?  Does the code centralize its approach? |
| **Authentication** | Look for weak passwords, clear-text credentials, overly long sessions, and other common authentication problems. |
| **Authorization** | Look for failure to limit database access, inadequate separation of privileges, and other common authorization problems. |
| **Sensitive data** | Does the code store secrets?  Is sensitive data stored in predictable locations? |
| **Unsafe code** | Is the code susceptible to buffer overruns?  Is the code susceptible to integer overflows?  Is the code susceptible to format string problems?  Is the code susceptible to array out-of-bound errors? |
| **Unmanaged code** | Does the code call potentially dangerous unmanaged APIs? |
| **Hard-coded secrets** | Look for hard-coded secrets in code by looking for variable names such as **"key"**, **"password"**, **"pwd"**, **"secret"**, **"hash"**, and **"salt"**. |
| **Poor error handling** | Does the code use proper and consistent error checking?  Does the application fail securely in the event of exceptions?  Do error messages give away too much information? |
| **Code that uses cryptography** | Did the team develop cryptographic algorithms?  Does the code use the right algorithm with an adequate key size?  Does the code generate random numbers for cryptographic purposes?  How does the code manage and store encryption keys?  How does the code manage and store encryption keys? |
| **Threading problems** | Is the code subject to race conditions?  Does the code contain static class constructors?  Does the code synchronize Dispose methods? |

**Execution Plan**

Secure Code Review activity will be carried out for all modules. The automated code scans and manual reviews (along with revalidation of previous reported issues) will be conducted post completion of every sprint/release.

After development of entire product and before deploying to production, another round of complete code review will be conducted.

### Web Application Security Assessment

Security testing team will perform controlled vulnerability assessment and exploitation attempts. The primary purpose of this testing is to identify and exploit application vulnerabilities present in the applications under scope. Following is the methodology that this assessment will follow.

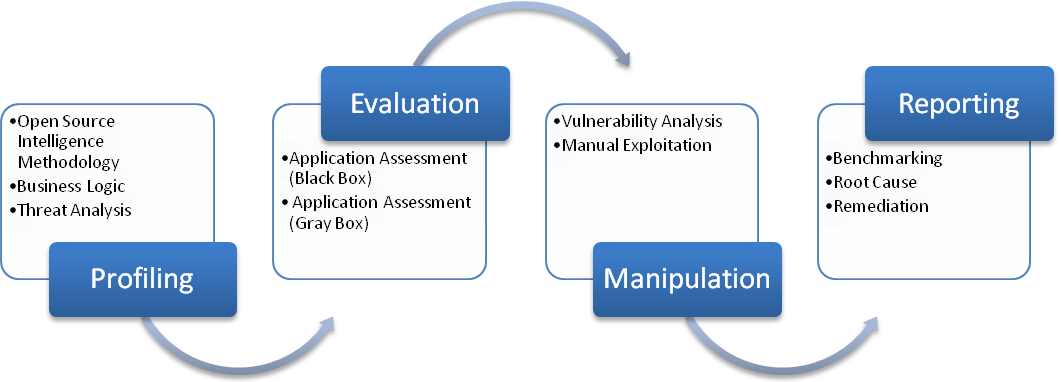


Figure 14 : Web Application security Assessment

#### Profiling

This phase consists of gathering information associated with the application and the related infrastructure. During this phase, consultants will observe and document the business logic and functionality of applications, this will include attempting to discover logic holes in the surface of the application. The business logic is the base of the application, and consultants want to determine if there are any cracks in the base that have not been previously discovered or known.

#### Evaluation

In this phase, the application is assessed using automated tools and manual techniques in order to identify Vulnerabilities present.

* During the automation phase, a controlled execution of automated tools is used to identify vulnerabilities that are presented to a user of the application in the form of an “anonymous user” and an “authorized user”. This allows a first round of identification of security issues in the application and provides a platform for manual testing.
* Black Box (as an anonymous user or an attacker) – Security consultants will not have any information about the application other than application URL and will try to get into the application without credentials, compromise system or retrieve sensitive information from the database.
* Gray Box (as an authorized user) – Security consultants will utilize the various credentials and will try to identify vulnerabilities which may allow an adversary to have privilege access of the application, compromise application or get unauthorized access to the database. The information required to carry out application assessment (Gray Box) is:
  + Application URL
  + Understanding of flow and functionality of the application
  + Test User Accounts Credentials of all types (e.g. Admin User, Regular User, Power Users etc.)
* The manual phase is where consultants use manual techniques to confirm the vulnerabilities found by the automated scanning.

#### Manipulation

In this phase, vulnerabilities/security observations identified in the previous phase is analyzed and/or exploited. Consultants will use an interception proxy to intercept and modify traffic to the application in order to exploit inherent weaknesses in the design and implementation of security controls in the application. This testing may involve performing three categories of tests which are:

* Input validation– Determine whether the application correctly filters user and application input. This includes the following:
  + Bounds checking (e.g. AAAAAAAA^16th input into forms)
  + Character Input (Unicode, URL encoding, HTML encoding attacks)
  + Form Bypass (Bypassing form stages that require information from a user)
* Manipulation testing – Determine the ability to bypass the application controls through manipulation of the user interfaces and application states. Tests include:
  + Session ID and state integrity
  + Parameters Manipulation
  + Page and form content manipulation
  + XSS testing
  + URL content manipulation
  + Injection attacks (e.g. SQL, LDAP)
* Structural Application Tests – Determine whether the application discloses information that could be inferred to launch further attacks. Tests include:
  + Elicit web server error messages
  + Database error messages
  + Directory and file content browsing
  + Legacy code searches
  + HTML comments

#### Reporting

The reporting phase of the assessment consists of aggregating all discovered and exploited vulnerabilities in a technical report that thoroughly describes risk, root cause, description of vulnerability, remediation steps, and links to vendor information on the vulnerability. The remediation steps are thoroughly communicated to the stakeholders during the reporting phase so that accepted vulnerabilities and vulnerabilities that cannot be mitigated are noted.

#### Execution Plan

Application Security Assessment (along with the underlying APIs) will be carried out for Pre-registration, REGISTRATION PROCESSOR, and Admin applications, after each module is fully functionally built and tested and deployment ready.

After development of entire product, another round of Application Security Assessment (along with revalidation of reported issues) will be carried out for the whole product, including integration of all modules, in the UAT environment. A separate (production replica) environment will be used for the testing, with relevant dummy data and accounts configured, where there will be no ongoing code or configuration changes.

### Web Service Security Assessment

Consultants perform security assessment on the Rest & SOAP based APIs with the objective of identifying and attempting to exploit the security vulnerabilities in them. The following methodology explains the steps followed in Web Service/API Security Assessment.

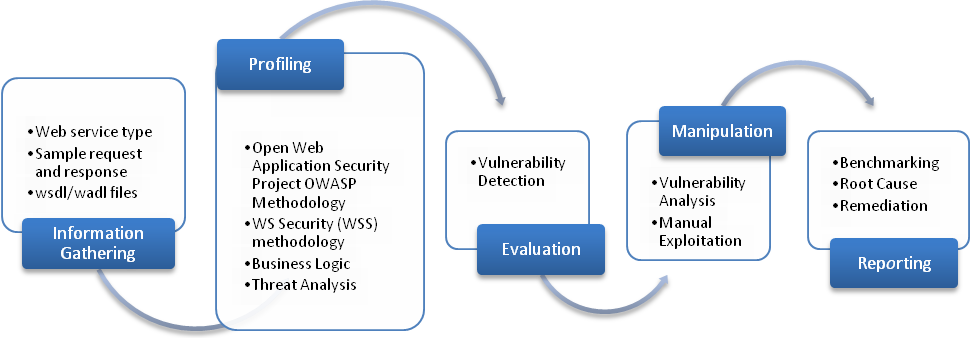


Figure 15 : Web Security Assessment

#### Information Gathering

This phase consists of collection of information which will be required for further analysis and security assessment. The minimum information required is as follows:

1. Type of Web Service used – REST or SOAP
2. Service description language – WADL/WSDL
3. Sample request and response for each end point – The sample request and response are collected for each API. A collection of valid and malformed requests is sent, and response are compared with valid and successful response. This helps in identifying successful exploits and vulnerabilities in an API.

#### Profiling

Well known industry standards are followed to create the test scenarios which will be the basis of the assessment in the upcoming phases. The business logic of the API is taken into consideration for identifying the threats.

#### Evaluation

In this phase, the API is assessed using automated tools and manual techniques in order to identify vulnerabilities present.

* During the automation phase, a controlled execution of automated tools is used to identify vulnerabilities that are present. This allows a first round of identification of security issues in the API and provides a platform for manual testing.
* The manual phase is where consultants use manual techniques to confirm the vulnerabilities found by the automated scanning.
* Black Box (as an anonymous user or an attacker) – Consultants will not have any information about the API other than URL and will try to exploit APIs without credentials, compromise system or retrieve sensitive information from the database.
* Gray Box (as an authorized user) – Consultants will utilize the various credentials and will try to identify vulnerabilities which may allow an adversary to have privilege access of the API, compromise API or get unauthorized access to the database.

#### Manipulation

In this phase, vulnerabilities/security observations identified in the previous phase is analyzed and/or exploited. Consultants will use an interception proxy to intercept and modify traffic in order to exploit inherent weaknesses in the design and implementation of security controls in the API. This testing may involve performing following categories of tests:

* Web Services Engine –
  + Buffer Overflows
  + XML parsing errors
  + Denial of Service
  + Large payload
* Web Services Deployment –
  + Session ID and state integrity
  + Parameters Manipulation
  + Injection and XSS attacks
* URL content manipulation –
  + Brute-force
  + Directory traversal
  + Information Leakage

#### Reporting

The reporting phase of the vulnerability assessment consists of aggregating all discovered and exploited vulnerabilities in a technical report that thoroughly describes risk, root cause, description of vulnerability, remediation steps, and links to vendor information on the vulnerability. The remediation steps are thoroughly communicated to the stakeholders during the reporting phase so that accepted vulnerabilities and vulnerabilities that cannot be mitigated are noted.

#### Execution Plan

Web Service Security Assessment will be carried out for all APIs of Registration, REGISTRATION PROCESSOR, IDS modules, after each module is built, tested and deployment ready.

After development of entire product, another round of Web Service Security Assessment (along with revalidation of reported issues) will be carried out for the whole product in the UAT environment. A separate (production replica) environment will be used for the testing, with relevant dummy data, API keys, etc., where there will be no ongoing code or configuration changes.

### Thick Client Security Assessment

A thick client (Fat Client) applications are heavy applications that normally entail the installation of application on the user computer/ client side. These types of application run on the computer’s memory and therefore the application’s security depends upon the local computer.

The approach to Thick Client Security Assessments take account of the review of data communication paths, server-side controls, and other client-related issues. Efforts are made to bypass the authentication controls, evaluate the data communications functionality, memory, file structure, registry and other forms that can make for the possible denial of service (DoS) attacks. Key attack vectors are:

* Network transmissions
* Client-side injection such as cross-site scripting and SQL injection
* Failure to implement the authorization policy
* Failure to protect key data/ resources
* Absence of data protection
* Insecure client-side storage
* Data storage such as Files, Windows registry, databases and other application’s executable files.

We use some of the following techniques to bypass the client-side validation for the correct evaluation of security aspects.

1. Man-in-The-Middle Attack
   1. Interrupt the client – server communication
   2. It does not require to understand the application code. It is one of the fastest way for security testing the application.
2. Reverse Engineer
   1. Identify the client – server communication code
   2. Disable the client-side validation
   3. It can be the time-consuming task and it depends upon the application technology.

The following are the list of security checks that will be performed on thick clients:

* Application profiling - This involves enumerating the application’s functionality and behavior, understanding the core security mechanisms employed by the application, identifying all of the different entry points for user input and establishing what technologies are being used on both the client and the server. This stage enables in determining the attack surface exposed by the application.
* Test the authentication mechanism - This involves testing the authentication-related functionality implemented within the application including registration, login, password change and account recovery functionality.
* Test the session management mechanism - This involves testing the mechanisms used for managing sessions and state including testing for insecure token generation and unsafe handling of tokens.
* Test access controls - This involves understanding the various access control requirements for the application and testing the implementation of access controls for defects leading to horizontal and vertical privilege escalation.
* Test the encryption control - This involves the testing for weak SSL/TLS ciphers suite usage, re-negotiation vulnerabilities, improper cryptography implementation, protection of private key and digital certificate related issues.
* Test for input-based vulnerabilities - This involves probing for input-based vulnerabilities that may arise anywhere within a typical application’s functionality, such as SQL injection, XSS, command injection and path traversal. It involves fuzzing every parameter to every request with a set of standard attack strings, and manually investigating all anomalous responses that may indicate the presence of vulnerability.
* Test for business logic flaws - This involves testing all relevant items of interesting functionality for logic flaws, including multi-stage processes, security-critical functions, etc.
* Test for sensitive data storage on files and registries - During the installation and execution of thick client applications, the thick clients might write/modify sensitive details in the files and registries. The sensitive data amassed by these applications usually contain username, passwords, database credentials, license details, cryptographic keys and configuration details like IP address, port, etc.
* Test for response modification - In the case of thick clients, most of the major processing/validations are carried at the client side. As a result, both the request as well as response modifications play a key role in testing the thick client for vulnerabilities.

**Execution Plan**

Thick client security assessment will be carried out for the registration client after the module is fully functional, tested and deployment ready.

After development of entire product, another round of Thick Client Security Assessment (along with revalidation of reported issues) will be carried out in the UAT environment. A separate (production replica) environment will be used for the testing, with relevant dummy data, user accounts, etc. where there will be no ongoing code or configuration changes.

# Functionalities Automatable and Not Automatable

**Following table details out the functionalities that are automated and those which are not automated:**

|  |  |  |
| --- | --- | --- |
| No. | Functionality | Mitigation |
|  | **Authenticity of the document been uploaded :** Content validation of Documents uploaded | Will be validated manually.  Only the allowed file types and the size will be validated automatically |
|  | **Autosuggestion** of the text fields | Easy to test manually. Will need to clear the cache entry the time and execute the tests twice |
|  | **Maps** and **geo-tagging** of the registration location | Will be validated manually under registration Admin.  A static/fixed data set will be used for automation. There is limited scope for automation as any data changes, these scripts need to be modified |
|  | Any **ACK receipt** sent to mobile as a **SMS** | Validation will be limited to the API layer and not with mobile network. Should be taken care as part of UAT testing |
|  | **Receipt** sent to Printer | Validation will be limited to the API layer and not with the physical devices. Should be taken care as part of UAT testing |
|  | **Capturing of Biometrics**  Finger print scanning/ Iris scanning will be by-passed with pre-set defined/captured test data | Will be validated manually. Should be taken care as part of UAT testing |
|  | **API Security** | Will not be part of automation, will be testing by a CEH by means of security test tools |
|  | **3rd Party Integration** Validation | Will not be part of automation.  Validation will be only limited to the API layer |
|  | **Integration** with **biometric** capturing devices of various manufacturers | Will be taken care as part of UAT testing |
|  | **Checking Biometrics Quality** as per Defined Standard/Threshold Value | Will be validated manually or part of UAT testing |
|  | **Interface with other COT products** like device managers, update servers, data providers | Will be taken care as part of UAT testing |

# Assumptions & Dependency

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Dependency on Item | Assumption considered | Impact/dependent on |
|  | **Secure Biometric Login** | All the Biometric data which shall be used as inputs to the Automated suite will be either in the **xml or Json format** where tags representing the binary values used for Authentication. | The Development team to provide the logic/program for conversion of biometric data set in the xml/json file which enfolds the Binary format of the biometric data set |
|  | **Biometrics - Support for Local Duplicate Checks** | The Automated suite will use a test data set of various combination of xml/json format packets for validation in the **Registration module having only the data of SO**  Registration processor system deduplication of biometric data will also be performed in a similar method | Deduplication validation of Biometrics will be tested by duplicate entries in Json/xml files |
|  | **Audit Logs** | Access to the log files of server to retrieve the data for validation to be made available from automation perspective. | Environment/Infra team |
|  | **UIN algorithm validation** | Conditions related to special numbers and duplicate numbers will be validated as the outcome from the algorithm. | Separate Scripts/Programs will be developed for testing the same |
|  | PKI keys for **decryption of packets** | The required keys for decryption of packets will be provided for the automation scripts to execute seamlessly | Development team |
|  | **API** / Webservices | All API’s and web services are documented and the endpoint URL are accessible for testing | Development team |
|  | **Localization** support | The Automation Framework will be designed to support localization and customization. The inputs of layouts/ configuration will be handled for country specific. However for maximum reusability of automated components **we assume the object identifiers for these layout/config won't override and will remain the same** | Development team |
|  | Primary **Language** | **English** will be considered as a primary language  The components used on the UI and their identifiers remains the same for multi-language support.  The framework will focus on languages written from Right to Left text editors (e.g. English) and will also, if needed later on incorporate support for Left to Right (e.g. Arabic) on need basis of the test scenarios | Requirements Gathering team |
|  | **Mobile** Application | There is no Mobile native application only mobile web application to be tested | Requirements Gathering team |
|  | **Automation Script Compatibility** | There is a high risk on reusability of test scriptsfor the scenarios that would be tested on Windows OS and Linux OS for the Registration module, which is a thick client application.  The OSS tools has limitation being an Image based recognition tools as the application being rendered on Windows OS and Linux OS may differ. | Once the Pilot app is ready a POC is recommended to be performed using Sikulix/TestFX for compatibility of automated scripts with Windows OS to see if it works/executes on Linux OS |
|  | **Registration Module to use Image based automation tools such as SikuliX or TestFX** | The overall automation coverage of the Registration client application which is a thick client may not be able to reach at least 60% as there could possibly be few technical limitations with the tool.  One of the major challenge foreseen is the objects will not get identified and there may be inconsistency with object recognition when the same scripts are run multiple times | There is no Open source tool available in the market which can support to the maximum extent for automation of Thick client application.  Where n ever the tests doesn’t give consistent results the tests would be executed in manual mode |
|  | **Test Environment** | A separate dedicated Pre-prod Environment (with data sets) will be made available for automation purposes | Infrastructure team |
|  | **Open source software** | The open source software’s identified and listed, their required packages are allowed to be installed on the environment the Automation team is supposed to work and the machine (desktop/laptop) will be enabled with Admin rights | Infrastructure team |
|  | **SMTP Server access** | Will be need access to setup a SMTP mail server to validate the Email sent by the system and the contents of the email as per the required specification | Infrastructure team |
|  | **Support** | The development team to provide sufficient support to resolve any potential issues. | Development team |

# Defect Management

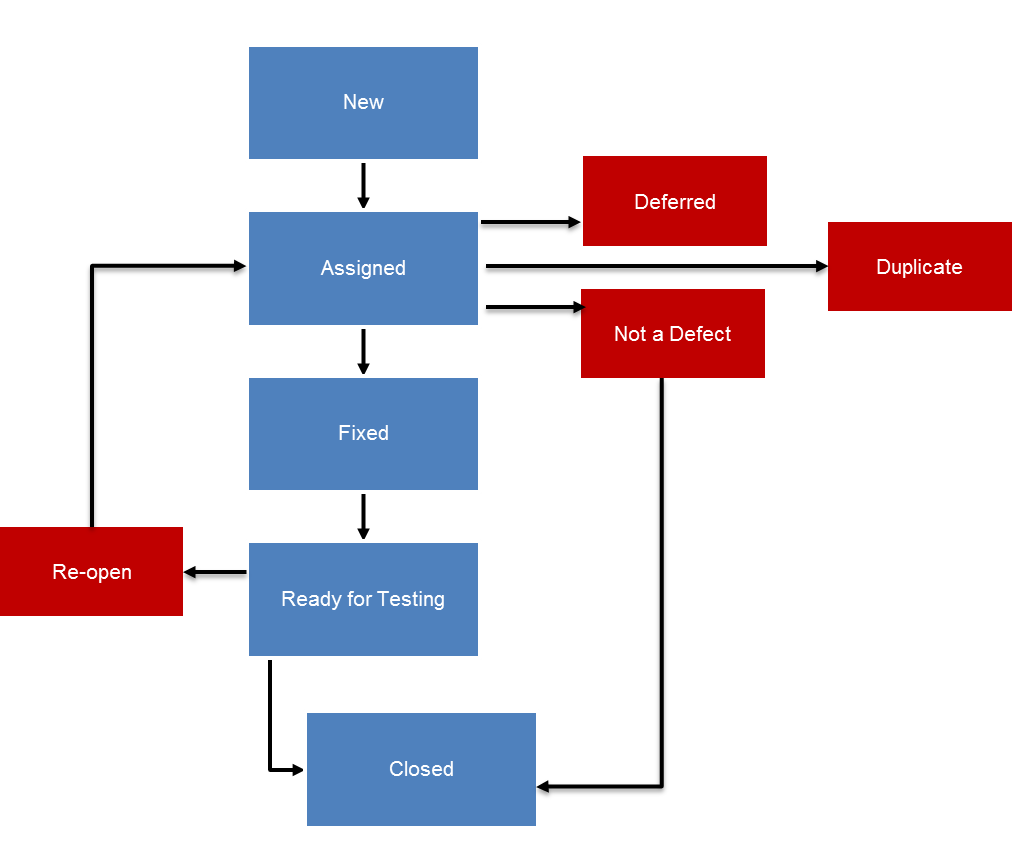
****

Figure 16 : Defect Cycle

|  |  |  |
| --- | --- | --- |
| **Defect status** | **Person responsible** | **Workflow in ALM** |
| New | Tester/ SDET | A tester/SDET will create a defect in ‘New state and assign it to the associated developer. Appropriate screenshots and detail defect description should be uploaded for each bug. |
| Assigned | Developer | Developer have to change the defect state from ‘New to ‘Assigned’ |
| Fixed | Developer | Developer to resolve the defect and change the defect state from ‘Assigned’ to ‘Fixed’ and assign the bug to Tester. The defect is still in ‘Development Sandbox’. |
| Not a Bug | Technical Architect/Dev Lead, Developer & respective Tester | Assigned developer change the defect state from ‘Assigned’ to ‘Not a Bug’ if the reported bug is not a valid bug. The defect should be assigned to Technical Architect/Dev Lead. Developer should provide appropriate comments.  Technical Architect/Dev Lead to analyze the bug and if found defect as invalid, the defect state will not be changed and assigned to Reporter of the issue (tester). Tester to verify and If tester agrees the defect is invalid then closes the defect. If Tester is not convinced, the defect to assign back to Technical Architect/Dev Lead and ask for clarification.  If the Technical Architect/Dev Lead find the defect is valid and rightly created, they should assign it back to developer and change defect state to ‘Assigned’ |
| Ready for  Testing | Developer | Post resolving the bug, the fix will be moved into Quality environment, dev team to change the defect state from ‘Fixed’ to ‘Ready for Testing’ and assign it to Reporter of the issue (tester). |
| Closed | Tester | Respective tester to verify the defect in ‘Ready for Testing’ and change the state to ‘Closed’ if the defect is found resolved. |
| Re-Open | Tester | Respective tester to change the Defect state from ‘Ready for Testing’ to ‘Re-Open’ and assign back to developer if the defect is not resolved with appropriate comments.  The defect state will be changed from ‘Not Fixed’ to ‘Assigned’. |
| Duplicate | Developer | Once the defect is assigned to the developer and if developer finds the defect is already logged / same or similar defect then the status will be changed to Duplicate with appropriate comments. |
| Deferred | Developer | The assigned defect is not possible to be fixed in the current sprint/release then post consulting the architect and SCM/ PO the developer will set the status to ‘Deferred’ These defects needs to be updated in backlog. |

While creating a defect it is necessary to assign severity to defect which will help to analyze the impact of the defect and appropriate action can be taken. Below table explain the guidelines which will be followed while assigning severity to the defect

| **Severity** | **Impact** |
| --- | --- |
| Sev1-Critical /Fatal  : System crashes, system is inoperable | * This bug is critical enough to crash the system, cause file corruption, or cause potential data loss * It causes an abnormal return to the operating system (crash or a system failure message appears). * It causes the application to hang and requires re-booting the system. * The defect affects critical functionality or critical data. It does not have a workaround. Example: Bio Deduplication is not rejecting the registration of a resident in-spite of sending the same data for a new registration |
| Sev2- Severe: Major function is inoperable, | * This Bug will degrade the quality of the System. However there is a workaround for achieving the desired functionality * This bug prevents other areas of the product from being tested. However other areas can be tested independently. * There are workarounds available |
| Sev3-Minor | * The defect does not affect functionality or data. * Minor function inoperable, workaround available   E.g. There is an insufficient or unclear error message in the set localized language , which has minimum impact on product use. |

# Testing Tool Stack

Testing Technology Stack

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Tool Name /Technology** | **Version** | **License** | **Remarks** |
| **Language** | Java | 1.8.x | Oracle Binary Code License |  |
| **IDE** | Eclipse | Oxygen package | Eclipse Public License | Java Integrated Development Environment |
| **Visual Design Test Case** | MindMap | 1.01 | GNU | Is a tool for visual flow representation |
| **Test Framework** | TestNG | 6.11 & above | Apache License 2.0 | is a testing framework for the Java programming language |
| **Tool** | Selenium | 2.53 & above | Licensed under the Apache 2.0 License | software-testing framework for web applications (it’s a Jar file) |
| **Tool** | Appium | 1.7.2 | Apache License 2.0 | tool for running scripts and testing native applications and mobile-web applications on android or iOS using a web driver |
| **Tool** | SikuliX | 1.1.2 | MIT License | GUI automation tool |
| **Library package** | Rest-assured | 3.0.0 | Apache License 2.0 | Testing and validating REST services in Java |
| **Library package** | Karate | 0.2.7 | MIT License | Rest API Test Tool Using Cucumber-JVM |
| **Library package** | WireMock | 2.16.0 & above | Apache License 2.0 | Simulator for HTTP-based APIs. (it’s a Jar file) |
| **Library package** | Citrus Framework | 2.7.4 | Apache License 2.0 | Simulate interface supporting Http, JMS, SOAP Web services, TCP/IP |
| **Tool** | JMeter | 4.x | Apache License 2.0 | Tool for web application, web services performance testing |
| **Tool** | TestFX |  | GNU | For client Application automation |
| **Tool** | Zabbix | 3.4 | GNU General Public License (GPL) version 2 | Tool for monitoring infrastructure during performance testing |
| **Test Management & Defect Tracking** | JIRA | 6.3.3 | Atlassian License | Tool for Test story tracking & defect management |
| **Scenario Management** | [Zephyr for Jira - Test Management](https://marketplace.atlassian.com/apps/1014681/zephyr-for-jira-test-management) | 3.6.3 |  | Scenario Management |
| **Security testing tools** | Burp suite Professional +  Acunetix, IBM Appscan or HP Web Inspect |  |  | Web/Mobile/Web service Application VA/PT |
| **Security Testing tools** | Fortify or Checkmarx or Veracode |  |  | Secure Code Review |

# Quality Gates

The path of evaluation happens at different phases in the testing life cycle qualifying the application from System / Integration Testing before moving to Production. Below are the entry, exit, suspension and resumption criteria for test execution.



### Entry Criteria

Testing Entry criteria:

* Feature for the release are approved and aligned
* Object Modeling process is completed for the desired features
* Approved Requirements & Acceptance criteria to be drafted by BA
* Unit Testing is done on the modules which are part of the test cycle
* QA build is deployed in QA Environment with no open critical defects
* QA Build has passed the sanity scenarios as part of the build process
* Required test data are available
* Test scenarios are developed, reviewed and approved
* Requirement traceability is achieved through the tool

### Exit Criteria

Test execution phase is considered as completed when following criteria are satisfied:

* 100% execution of the feature list built on the features designed for the release
* Agreed functionality have been delivered (Client Valued functionality)
* All agreed defects are fixed, re-tested and closed. In the event of inability to fix a defect, the defect will go through review for MOSIP Teams approval.
* Sign off to be given by the stakeholders on the testing deliverables
* Known issues are to be aligned with the stakeholders

### Suspension Criteria

The Testing will be suspended under the following circumstances and decision should be made in consensus with Manager/ stakeholder:

* The agreed test feature components are not available for testing
* A critical defect have been identified which impacts the testing progress
* Failure in smoke testing on the build deployed into the test environment
* Non availability of needed test data / test environment

### Resumption Criteria

The testing team will be resuming testing under following circumstances and decision will be made in consensus with the project manager/ stakeholder

* The cause of the suspension has been resolved
* The application development / build team has resolved the cause of the suspension and has certified that system is stable enough to continue testing
* All the needed setup / environment is provided with a stable build which will help in resuming the formal testing

# RASCI matrix

RASCI matrix describes participation of various QA roles in completing project tasks. It is useful for clarifying roles and responsibilities of different QA team members.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Phase** | **Testing/RASCI Activities** |  |  |
|  | **R - Responsible - owns the problem / project** | **Test In charge** | **Test Engineer (SDET Manual/automation/ performance)** |
| **A - to whom "R" is Accountable - who must sign off (Approve) on work before it is effective** |
| **S - Supportive - can provide resources of can play a supporting role in implementation** |
| **C - to be consulted - has information and / or capacity necessary to complete the work** |
| **I - to be informed - must be notified of results, but need not be consulted** |
| **Test Planning** | Requirement Understanding | R | R |
| Define the Test Strategy for different phases of testing | A/R | I |
| Test Plan creation | A | S |
| Identify the Testing tools & Frameworks | A | R |
| **Test Preparation** | Test Scenario & Test script creation | A | R |
| Test Scenario & Test script review | A/R | S |
| Test Environment and Data set-up | C | R |
| Define the Frameworks | A | S |
| **Test Execution** | Functional & Automation Test Execution | A | R |
| System Integration Test Execution | A | R |
| Usability & Compatibility Test Execution | A | R |
| Test Co-ordination (Integration team & External teams) | A | R |
| Defect Reporting | A | R |
| Defect Retesting & Regression Test Execution | A | R |
| Test Summary Report - Preparation & Review | A/R | R/S |
| Defect Summary Report (Closed / Deferred defects) | A/R | R/ S |
| Metrics- Capturing & Analysis | A/R | R/S |

# Test Deliverables

The transparency on the QA activities during the test cycle is important for all stakeholders. Hence reporting becomes critical. Below are the sample reports which are part of the deliverables.

|  |  |  |
| --- | --- | --- |
| **Name of the Report** | **Data to be provided** | **Significance** |
| Story Test Report | This report will be available in JIRA for test execution against each Story. Email will be shared with all stakeholders once the story is tested by QA. This will be used for sign off by QA. | Story testing status & sign off story to UAT |
| Release/Test Summary Report | Overall summary of QA activities, its status and QA metrics. | Quality of release |
| MindMap Diagrams | MindMap images covering the user stories. | Sign off from the QA covering the flows of the User story |
| Automation Scripts | Automation scripts covering the user stories | Sign off from the QA covering the flows of the User story |
| Test Execution report | Test case execution report. This can be used for regression test execution also. | Publish the test execution results & highlight risks if any |
| Automation test execution report | Test results of automation testing on QA or UAT environment | Overall build quality |
| Performance test execution report | Test results of performance testing | Overall performance of the application |
| Weekly Status Report | Weekly QA status summary | Status update |
| Daily Status Report | Activity summary and plan for next day | Status update |
| Monthly dashboard | Summary of monthly activities, accomplishments, issues etc | Status update |

# Test metrics

Tracking of KPI is very important to ensure that quality of deliverables maintained and improved over the period of time. Below table shows the metrics which are recommended for MOSIP

### Functional Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KPI | Metric | Purpose | Frequency | Details |
| Productivity | % completion | Track productivity of QA team | End of Release | = # features list for which testing is complete / # feature list received during release |
| Test scenario executed Vs Planned (Manual and automation) | Measure the progress of planned Vs Actual test cases executed | End of sprint/Release | =Number of test scenarios executed/ Number of test scenarios planned |
| Automation test coverage | Measure automation test coverage | End of release | = Number of test scenario automated / Total number of test scenarios those can be automated |
| Test design effectiveness | Test scenario effectiveness | To measure the effectiveness of TS | End of Release | =Number of defects identified by TS/Total number of TS executed |
| % Adhoc defect | To measure the effectiveness of TC and coverage. More number of Adhoc defects indicates Test case coverage needs to be improved | End of Release | = Adhoc defects/Total defects |
| Quality of deliverables | Defect Removal Efficiency | To measure testing quality | End of release | = Total number of defects / (Total number of internal defects + Total number of UAT + Production defects\*) |
| Defect Density | To measure code quality | End of Release | = Number of defects detected/ Total size of QA scenarios |
| Defect leakage | To measure testing quality | End of Subsequent release | Number of defects logged in production\* |
| Defect rejection ratio | To measure testing quality | End of Release | Number of invalid defects/Total number of defects |
| Reopened defects | To measure quality of defect fixes | End of Release | Number of defects reopened |
| Defects by Severity | To measure quality of code and testing quality | End of Release | Number of defects for each level of severity |
| Cost | Burn down chart (Effort variance) | Tracks the actual effort, against the committed effort over release. | End of Release | (# of Actual Hours - # of Hours Committed) / # of Hours Committed |
| Automation ROI | Return on investment from Automation | Helps assess the breakeven for the automation cost/effort | End of Release | Analysis of indicative cycles when the investment will be yield savings |

### Performance Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KPI** | **Metric** | **Purpose** | **Frequency** | **Details** |
| Response time | Response time in seconds / milliseconds | Measure responsiveness of key transactions of the application | For each performance test  For example: |  |
| 90th Percentile transaction response time | Response time in seconds / milliseconds | Maximum response time for ninety percent of the transactions (in seconds) |  |  |
| Throughput | # of requests / second | Measure # of completed requests per unit time to understand capacity of application | For each performance test |  |
| Error Rate | % of requests failed against total requests send | Measure # of failed requests during tests | For each performance test |  |
| CPU Utilization | Overall % of CPU utilized for the server | Measure overall CPU utilization while application is subjected to load | During each performance test |  |
| Memory Utilization | Available Mbytes | Available Mbytes while application is subjected to load | For each performance test |  |
| Network Utilization | Bytes Received / sec  Bytes sent / sec | Byes Sent and Received per unit time to understand how network bandwidth is used | For each performance test |  |
| Disk Utilization | Disk Reads / sec  Disk Writes/ sec | Disk Read and Write statistics to measure disk capacity | For each performance test |  |

### Security Testing Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KPI** | **Metric** | **Purpose** | **Frequency** | **Details** |
| **Security** | Number of Vulnerabilities Identified | Measure the effective security of the application | End of security testing | Number of vulnerabilities identified and their relevant category such as Business logic, Injection flaw, etc. |
| Vulnerability Severity | To measure security risk  - critical, high, medium and low | End of security testing and revalidation activity | Helps in analyzing the risk and prioritizing the vulnerabilities.  Allows for defining policy on vulnerability mitigation timelines such as: critical to be fixed within 1-2 days, high within a week, medium within a month and low within a quarter. |

**\*KPI’s considered for all the defects detected during the warranty period are against the MOISP platform only**